01_en

May 20, 2022

0.1 Introduction

0.1.1 Basic programming concepts

- **Algorithm**: Finite sequence of steps for solving a specific problem.
- The concept can be applied for everyday tasks too (e.g. making a Sacher cake, cleaning the bookshelf :-).
- **Data structure**: A scheme for the storage and efficient use of data elements (example: list).
- **Programming language**: A language defined by rigorous rules, that can be used to communicate instructions to the computer.
- **Programming**: Designing algorithms and data structures, and implementing them in a programming language (coding).

0.1.2 Characteristics of the Python programming language

+ concise and elegant syntax + easy to learn ("brain-friendly") + tens of thousands of external packages (https://pypi.org/) + strong community, annual PyCon conferences + free and open source + platform independent + dynamically typed, interpreted language + multi-paradigm language - can be slow for certain tasks - its multi-threaded capabilities are limited

0.1.3 History

- **1994**: Python 1.0 was released.
- **2000**: Python 2.0 was released.
- **2001**: The Python Software Foundation was launched.
- **2003**: The first PyCon conference was held.
- **2008**: Python 3.0 was released. It was not compatible with version 2. The transition was slow, but finally happened.
- **2018**: Guido van Rossum steps down as BDFL. The main authority behind the language will be a five-person steering committee (see: PEP 8016).

0.2 The Jupyter Notebook environment

- Jupyter Notebook is a browser based, interactive work environment.
- It was developed primarily for Python, but it can be used with other programming languages too.

- A notebook consists of cells, the cell type can be text (Markdown) or code.
- Code cells can be executed, even multiple times. The result of the execution is displayed after the given code cell.
- A notebook can be used in two modes:
- In command mode, we can perform cell level operations (e.g. insert new cell, delete cell, move cells, switch between the cells, etc). A few keyboard shortcuts:
  - `b`: Insert new code cell below the current cell.
  - `m`: Change the type of the current cell to text.
  - `dd`: Delete the current cell.
  - `Enter`: Switch to edit mode (to edit the content of the current cell).
- In edit mode, we can edit the content of cells. A few keyboard shortcuts:
  - `Ctrl+Enter`: Run the actual cell.
  - `Esc`: Toggle back to command mode.
- For a detailed description of keyboard shortcuts, see the menu item Help / Keyboard Shortcuts.

0.3 Simple data types

0.3.1 Integer number

[1]:
```
# We can perform operations between numbers the usual way.
1 + 1

# Remarks:
# - Whitespaces do not count, the formatting of the above code follows the PEP␣
 ↪8 coding style.
# - Jupyter displays the last expression of the cell after the execution.
```

[1]:
```
2
```

[2]:
```
# To override the default precedence, use parentheses!
(2 + 3) * 4
```

[2]:
```
20
```

[3]:
```
# Python can handle arbitrarily long integers, there is no overflow error.
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
 ↪+ 1
```

[3]:
```
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111112
```

[4]:
```
# Create a variable called i, and assign the value 11 to it!
i = 11

# Remarks:
```

```
# - = is the symbol of the assignment operation.
# - i gets the specified value, but the assignment itself provides no result.␣
  ↪Therefore, the cell has no output.
```

[5]:
```
# The variable can be used in further expressions.
2 * i
```

[5]: 22

[6]:
```
# Of course, the value of the variable can be changed.
i = 42
i
```

[6]: 42

[7]:
```
# Assignment can be combined with other operations.
i += 1 # equivalent with i = i + 1
i
```

[7]: 43

[8]:
```
# Floating point division.
7 / 3
```

[8]: 2.3333333333333335

[9]:
```
# Integer division (the fractional part is discarded). It prevents many errors␣
  ↪that it has a separate symbol.
7 // 3
```

[9]: 2

[10]:
```
# Modulo operation (calculates the remainder).
7 % 3
```

[10]: 1

[11]:
```
# There is also a power operation, denoted by **.
2**10
```

[11]: 1024

0.3.2 Floating point number

- Floating point arithmetic makes it possible to perform approximate calculations with real numbers.

- Python's floating point type implements the (64 bit) double precision type of the IEEE-754 standard.

[12]:
```
# Floating point constants can be given using the decimal point.
1.23 * 4.56
```

[12]:
```
5.6088
```

[13]:
```
# Calculating the square root of two (approximately).
2**0.5
```

[13]:
```
1.4142135623730951
```

[14]:
```
# Create a float variable called f!
f = 1.5
f
```

[14]:
```
1.5
```

[15]:
```
# The type of f can be queried with the type() function.
type(f)
```

[15]:
```
float
```

[17]:
```
# ...type() also works on any other value.
type(2 * 3)
```

[17]:
```
int
```

[18]:
```
# Now let's put an integer value into f!
# In Python, this can be done without any problem.
f = 100
type(f)
```

[18]:
```
int
```

0.3.3 Complex number

- Python supports complex numbers, without the need for external libraries.

[18]:
```
# Division in algebraic form.
(2 + 3j) / (5 - 3.5j)
```

[18]:
```
(-0.013422818791946262+0.5906040268456377j)
```

[19]:
```
# Raising the imaginary unit to a power.
1j**2019
```

[19]: (8.77583695147045e-14-1j)

0.3.4 String

- The string data type is used to store text values.
- In Python, a string is an immutable sequence of Unicode symbols (or Unicode characters).

[21]:
```
# String constants are delimited by ' signs.
'apple'
```

[21]: 'apple'

[23]:
```
# ...but " signs can also be used.
"pear"
```

[23]: 'pear'

[24]:
```
# Remark: In the output of the previous cells, ' is not part of the string, it␣
↪only indicates the data type.
# Let's print the content of the string, without the delimiters!
print('apple')
```

```
apple
```

[28]:
```
# The type() function works in this case too.
type('apple')
```

[28]: str

[24]:
```
# Of course we can use Unicode symbols in the string.
'I ♥ '
```

[24]: 'I ♥ '

[25]:
```
# The rationale of the two delimiter signs:
print("asd'fgh")
print('asd"fgh')
```

```
asd'fgh
asd"fgh
```

[26]:
```
# ...otherwise the ' and " character should be escaped.
print('asd\'fgh')
print("asd\"fgh")
```

```
asd'fgh
asd"fgh
```

[27]:
```
# Create a string variable called s!
s = 'beer'
```

[30]:
```
s
```

[30]:
```
'beer'
```

[29]:
```
# Extracting the characters of s.
# Remark: Indexing starts from 0.
s[0]
```

[29]:
```
'b'
```

[31]:
```
# The character is returned as a string of length 1.
type(s[0])
```

[31]:
```
str
```

[33]:
```
# If the index is too large, then we get an error message.
s[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-33-ae2be691b535> in <module>
      1 # If the index is too large, then we get an error message.
----> 2 s[4]

IndexError: string index out of range
```

[34]:
```
# The characters of the string cannot be modified!
# (We will see later, why.)
s[0] = 'x'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-774b7fe166a1> in <module>
      1 # The characters of the string cannot be modified!
      2 # (We will see later, why.)
----> 3 s[0] = 'x'

TypeError: 'str' object does not support item assignment
```

[35]:
```
# Of course, the variable s can get a new value.
s = 'wine'
```

```
# Remark: The assignment is executed, but the assignment expression itself has␣
 ↪no result.
# Therefore the cell has no output.
```

[36]:
```
# Let's print the content of s!
print(s)
```

```
wine
```

[37]:
```
# The length of the string (number of Unicode symbols):
len('John ')
```

[37]: 5

[38]:
```
# Concatenating strings.
'beer' + 'wine'
```

[38]: 'beerwine'

[40]:
```
# Is a string contained in another one?
'ab' in 'abracadabra'
```

[40]: True

[42]:
```
'xyz' in 'abracadabra'
```

[42]: False

[43]:
```
# A string can be encoded into a byte sequence (using an encoding scheme).
b = 'Géza'.encode('utf-8')
b
```

[43]: b'G\xc3\xa9za'

[45]:
```
# Type of the result.
type(b)
```

[45]: bytes

[46]:
```
# The number of bytes can be greater than the number of Unicode symbols!
len(b)
```

[46]: 5

[49]:
```
# Exercise:
# How long is the UTF-8 representation of the lowercase accented letters in the␣
 ↪Hungarian alphabet?
```

```
print(len('á'.encode('utf-8')))
print(len('é'.encode('utf-8')))
print(len('í'.encode('utf-8')))
print(len('ó'.encode('utf-8')))
print(len('ö'.encode('utf-8')))
print(len('ő'.encode('utf-8')))
print(len('ú'.encode('utf-8')))
print(len('ü'.encode('utf-8')))
print(len('ű'.encode('utf-8')))

# Remark: The code above is full of repetitions. We will learn soon, how can it␣
  ↪be made more elegant.
```

```
2
2
2
2
2
2
2
2
2
```

[50]:
```
# How long is the UTF-8 representation of   and  ?
print(len(' '.encode('utf-8')))
print(len(' '.encode('utf-8')))
```

```
3
3
```

[51]:
```
# The operation that transforms a byte sequence into a string is called␣
  ↪decoding.
b.decode('utf-8')
```

[51]: 'Géza'

[52]:
```
# Creating an empty string.
''
```

[52]: ''

[53]:
```
len('')
```

[53]: 0

[54]:
```
# Remove leading and trailing whitespace (space, tab, line break)
# characters from the string.
'  \tapple\n'.strip()
```

[54]: 'apple'

[62]:
```
# Remove a specified set of characters instead.
'---alma+++'.strip('+-')
```

[62]: 'alma'

[69]:
```
# Convert to lowercase.
'Foo'.lower()
```

[69]: 'foo'

[68]:
```
# Convert to uppercase.
'bar'.upper()
```

[68]: 'BAR'

[67]:
```
# Repeat string the given number of times.
'ma' * 4
```

[67]: 'mamamama'

0.3.5 Boolean value

- This type represents the truth values of Boolean logic as *True* and *False*. Capital initials are important as Python is case sensitive.

[50]:
```
# Create a boolean variable!
x = True
x
```

[50]: True

[55]:
```
# Logical AND operation.
print(True and False)
print(True and True)
```

```
False
True
```

[56]:
```
# Logical OR operation.
print(False or False)
print(False or True)
```

```
False
True
```

[53]:
```
# Logical negation.
not x
```

[53]:
```
False
```

[57]:
```
# The result of a comparison operation is a logical value.
print(2 <= 3)
print(5 > 10)
```

```
True
False
```

[59]:
```
# The sign of equality testing is ==.
print('apple' == 'apple')
print('apple' == 'pear')
```

```
True
False
```

0.3.6 None

- In Python, None values have a placeholder role. For example, None can indicate missing or invalid result, or the default setting.

[60]:
```
# Type of the value None.
type(None)
```

[60]:
```
NoneType
```

[61]:
```
# If the cell's last expression is None, then there is no output.
1 + 1
None
```

02_en

May 20, 2022

0.1 Collections

0.1.1 Tuple

- A tuple is an immutable array that can be indexed by natural numbers.
- The items of the tuple do not need to have the same type.
- Indexing takes O(1), inclusion test takes O(n) time, where n is the number of items.

[1]:
```
# Create a tuple variable t that contains 3 items!
t = (10, 20, 30)
t
```

[1]:
```
(10, 20, 30)
```

[2]:
```
# Check the type of t!
type(t)
```

[2]:
```
tuple
```

[3]:
```
# The len() function returns the number of items.
len(t)
```

[3]:
```
3
```

[4]:
```
# Accessing an item of the tuple (indexing starts from 0).
t[1]
```

[4]:
```
20
```

[5]:
```
# The items cannot be modified!
t[1] = 200
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-194e2b48bb60> in <module>
      1 # The items cannot be modified!
----> 2 t[1] = 200
```

```
TypeError: 'tuple' object does not support item assignment
```

[6]:
```
# However, the variable t can get a new value.
t = (30, 40)
t
```

[6]:
```
(30, 40)
```

[7]:
```
# The items do not need to have the same type.
t = (20, 1.5, 'foo', (100, 200))
```

[8]:
```
# Inclusion test.
1.5 in t
```

[8]:
```
True
```

[9]:
```
100 in t
```

[9]:
```
False
```

[10]:
```
# If it does not cause ambiguity, the ( and ) signs can be omitted!
t = 50, 60, 70
type(t)
```

[10]:
```
tuple
```

[11]:
```
# Creating an empty tuple.
()
```

[11]:
```
()
```

[12]:
```
# Creating a tuple of size one.
(42,)
```

[12]:
```
(42,)
```

0.1.2 List

- The modifiable version of tuple. New items can be added, and existing items can be changed too.
- Indexing takes O(1), inclusion test takes O(n) time, like for tuples.

[13]:
```
# Create a list variable l that has 4 items!
# The items do not need to have the same type.
l = [2, 3, 'wine', 5]
```

[14]:
```
# Check the type of l and query the number of items!
type(l), len(l)
```

[14]:
```
(list, 4)
```

[15]:
```
# Accessing the items of the list (indexing starts from 0).
l[2]
```

[15]:
```
'wine'
```

[16]:
```
# Modifying list item.
l[2] = 'beer'
l
```

[16]:
```
[2, 3, 'beer', 5]
```

[17]:
```
# A list can contain another list as an item.
[3, 5, [10, 20]]
```

[17]:
```
[3, 5, [10, 20]]
```

[18]:
```
# Append item to the end of the list.
l.append(100)
l
```

[18]:
```
[2, 3, 'beer', 5, 100]
```

[19]:
```
# Insert item into the middle of the list.
l.insert(2, 'apple')
l
```

[19]:
```
[2, 3, 'apple', 'beer', 5, 100]
```

[20]:
```
# Inclusion test.
'beer' in l
```

[20]:
```
True
```

[21]:
```
# Determine the index of a given value (the first occurrence).
l.index(5)
```

[21]:
```
4
```

[22]:
```
# Append all items of a sequence to the list.
l = [10, 10, 20]
l.extend([30, 40])
l
```

[22]: [10, 10, 20, 30, 40]

[23]:
```
# The extend() function is different from append()!
l = [10, 10, 20]
l.append([30, 40])
l
```

[23]: [10, 10, 20, [30, 40]]

[24]:
```
# Remove item at the given index.
l = [10, 20, 30, 40]
l.pop(1)
```

[24]: 20

[25]:
```
l
```

[25]: [10, 30, 40]

[26]:
```
# Remove last item.
l.pop()
```

[26]: 40

[27]:
```
l
```

[27]: [10, 30]

[28]:
```
# Concatenate two lists.
[20, 30, 40] + ['apple', 'pear']
```

[28]: [20, 30, 40, 'apple', 'pear']

[29]:
```
# Repeat list the specified number of times.
['left', 'right'] * 3
```

[29]: ['left', 'right', 'left', 'right', 'left', 'right']

[30]:
```
# Create empty list.
[]
```

[30]: []

0.1.3 Set

- The set data type is an implementation of the mathematical concept of set.
- A set cannot be indexed, inclusion test runs in O(1) time.

[31]:
```
# Create a set variable s!
s = {10, 20, 30}
```

[32]:
```
# Check the type and element count of s!
type(s), len(s)
```

[32]: (set, 3)

[33]:
```
# Inclusion test.
30 in s
```

[33]: True

[34]:
```
# Adding an element to the set.
s.add(42)
s
```

[34]: {10, 20, 30, 42}

[35]:
```
s.add(42)
s
```

[35]: {10, 20, 30, 42}

[36]:
```
# Set operations.
{1, 2, 3, 4} | {3, 4, 5} # union
```

[36]: {1, 2, 3, 4, 5}

[37]:
```
{1, 2, 3, 4} & {3, 4, 5} # intersection
```

[37]: {3, 4}

[38]:
```
{1, 2, 3, 4} - {3, 4, 5} # set difference
```

[38]: {1, 2}

[39]:
```
# The type of the elements is not necessarily identical.
{1, 2, 'Robin Hood'}
```

[39]: {1, 2, 'Robin Hood'}

[40]:
```
# A set can contain any element of immutable type.
{10, 2.5, 'apple', (20, 30)}
```

[40]: {(20, 30), 10, 2.5, 'apple'}

[41]:
```
# ...but it cannot contain a mutable element!
{10, [20, 30]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-011030676959> in <module>
      1 # …but it cannot contain a mutable element!
----> 2 {10, [20, 30]}

TypeError: unhashable type: 'list'
```

[42]:
```
# Removing an element.
s = {10, 20, 30}
s.remove(20)
s
```

[42]:
```
{10, 30}
```

[43]:
```
# Creating an empty set.
set()
```

[43]:
```
set()
```

0.1.4 Dictionary

- A dict(ionary) is a set of key-value pairs, where the keys are unique.
- The key's data type can be a simple data type, tuple or any immutable data type.
- Indexing can be done by the key, in O(1) time.

[44]:
```
# Create a dict variable d!
d = {'apple': 10, 'pear': 20, 2.5: 100}
```

[45]:
```
# Check the type and the element count of d!
type(d), len(d)
```

[45]:
```
(dict, 3)
```

[46]:
```
# Querying the assigned value of an existing key.
d['apple']
```

[46]:
```
10
```

[47]:
```
# Querying the assigned value of a non-existent key.
d['Johnny']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-47-0419045f4826> in <module>
      1 # Querying the assigned value of a non-existent key.
----> 2 d['Johnny']

KeyError: 'Johnny'
```

[48]:
```
# Changing the assigned value of a key.
d['apple'] = 42
d
```

[48]:
```
{'apple': 42, 'pear': 20, 2.5: 100}
```

[49]:
```
# Inserting a new key-value pair.
d['cherry'] = [40, 50]
d
```

[49]:
```
{'apple': 42, 'pear': 20, 2.5: 100, 'cherry': [40, 50]}
```

[50]:
```
# Removing a key-value pair.
del d['apple']
d
```

[50]:
```
{'pear': 20, 2.5: 100, 'cherry': [40, 50]}
```

[51]:
```
# Is a key contained in the dict?
'pear' in d
```

[51]:
```
True
```

[52]:
```
'xx' in d
```

[52]:
```
False
```

[53]:
```
# Creating an empty dict.
{}
```

[53]:
```
{}
```

[54]:
```
# Even tuples can be dict keys.
x = {(1, 2): "qq", (3, 4): 'ww'}
x
```

[54]:
```
{(1, 2): 'qq', (3, 4): 'ww'}
```

[55]:
```
x[(1, 2)]
```

[55]:
```
'qq'
```

[56]:
```
x[1, 2]
```

[56]:
```
'qq'
```

0.2 Conversion

For every data type discussed so far, there is a function that converts to the given type from any other type, if the conversion makes sense.

[57]:
```
int(2.7) # float => int
```

[57]:
```
2
```

[58]:
```
int('12') # str => int
```

[58]:
```
12
```

[59]:
```
float('10') # str => float
```

[59]:
```
10.0
```

[60]:
```
float(42) # int => float
```

[60]:
```
42.0
```

[61]:
```
str(20) # int => str
```

[61]:
```
'20'
```

[62]:
```
tuple([1, 2, 3]) # list => tuple
```

[62]:
```
(1, 2, 3)
```

[63]:
```
list((4, 5, 6)) # tuple => list
```

[63]:
```
[4, 5, 6]
```

[64]:
```
set((7, 8, 9)) # tuple => set
```

[64]:
```
{7, 8, 9}
```

[65]:
```
dict([('a', 1), ('b', 2)]) # list of pairs => dict
```

[65]:
```
{'a': 1, 'b': 2}
```

[66]:
```
list({'a': 1, 'b': 2}.items()) # dict => list of pairs
```

[66]:
```
[('a', 1), ('b', 2)]
```

[67]:
```
# Creating an empty list (alternative solution).
list()
```

[67]:
```
[]
```

03_en

May 20, 2022

0.1 Standard streams

The operating system assigns 3 standard streams to each process at startup: standard input, standard output and standard error. By default, standard input is connected to the keyboard, standard output and error are connected to the display. This setting can be changed, e.g. the standard input can come from a file or another program, the standard output can be directed to a file or another program.

0.1.1 Standard input

- We can read from the standard input using the input function.
- The type of the result is string. Conversion is needed to obtain a different data type.

[1]:
```
# Read input of type string.
x = input('Type some text: ')
x
```

```
Type some text: This is great!
```

[1]:
```
'This is great!'
```

[2]:
```
# Read integer input.
y = int(input('Type an integer number: '))
y
```

```
Type an integer number: 42
```

[2]:
```
42
```

0.1.2 Standard output and error

- We can write to the standard output and error using the print function.

[3]:
```
# Printing to the standard output.
print('hello')
print('bello')
```

```
hello
bello
```

[6]:
```
# Printing into the same line.
print('hello', end='')
print('bello', end='')
```

```
hellobello
```

[15]:
```
# Printing a line break.
print()
```

[7]:
```
# Printing to standard error.
import sys
print('ERROR!', file=sys.stderr)
```

```
ERROR!
```

0.1.3 Output formatting

[8]:
```
# Output formatting using an f-string.
x1 = 2.5
x2 = 4.78
print(f'The first solution is {x1}, the second solution is {x2}.')
```

```
The first solution is 2.5, the second solution is 4.78.
```

[9]:
```
# Printing with 1 decimal place accuracy.
print(f'The first solution is {x1:.1f}, the second solution is {x2:.1f}.')
```

```
The first solution is 2.5, the second solution is 4.8.
```

[10]:
```
# Printing an integer and a string.
i = 42
s = 'apple'
print(f'integer: {i}, string: {s}')
```

```
integer: 42, string: apple
```

[11]:
```
# Output formatting using the % operator.
x1 = 2.5
x2 = 4.78
print('The first solution is %f, the second solution is: %f.' % (x1, x2))
```

```
The first solution is 2.500000, the second solution is: 4.780000.
```

[12]:
```
# Printing with 1 decimal place accuracy.
print('The first solution is %.1f, the second solution is %.1f.' % (x1, x2))
```

```
The first solution is 2.5, the second solution is 4.8.
```

[13]:
```
# Printing an integer and a string.
i = 42
s = 'apple'
print('integer: %d, string: %s' % (i, s))
```

```
integer: 42, string: apple
```

[14]:
```
# Remark: A f-strings and the % operator
# can be used without printing, as a string operation.
f'One plus one is {1 + 1}.'
```

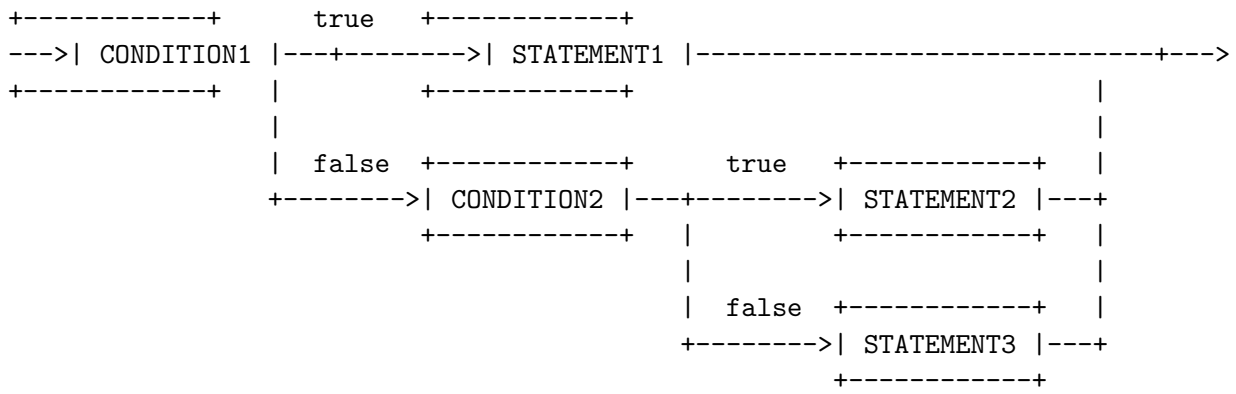
[14]:
```
'One plus one is 2.'
```

0.2 Control structures

- In Python, the body of control structures is marked by indentation.
- Therefore, the visual appearance of a program is always consistent with its logical meaning.

0.2.1 if statement

- Syntax: “‘ if CONDITION1: STATEMENT1 elif CONDITION2: STATEMENT2 else: STATEMENT3
- Flow diagram:



- Remarks:
- There can be multiple elif branches.
- The elif branche and the else branch can be omitted.
- If the statement is 1 line long, then it can be written into the same line with if, elif and else.

[19]:
```
# Example: Do you want a beer?
x = int(input('How old are you? '))
if x >= 18:
    print('Do you want a beer?')
else:
    print('Cannot serve beer to you.')
```

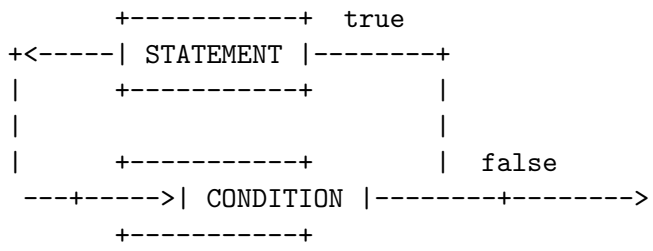
```
How old are you? 20
Do you want a beer?
```

[18]:
```
# Example: Quadratic equation solver.

# get a, b and c
a = float(input('a: '))
b = float(input('b: '))
c = float(input('c: '))

# compute d
d = b**2 - 4 * a * c

if d > 0:    # 2 solutions
    x1 = (-b + d**0.5) / (2 * a)
    x2 = (-b - d**0.5) / (2 * a)
    print(f'x1={x1}, x2={x2}')
elif d == 0: # 1 solution
    x1 = -b / (2 * a)
    print(f'x1={x1}')
else:        # 0 solutions
    print('No solution!')
```

```
a: 1
b: 3
c: 2
x1=-1.0, x2=-2.0
```

0.2.2 while statement

- Syntax: “‘ while CONDITION: STATEMENT
- Flow diagram:



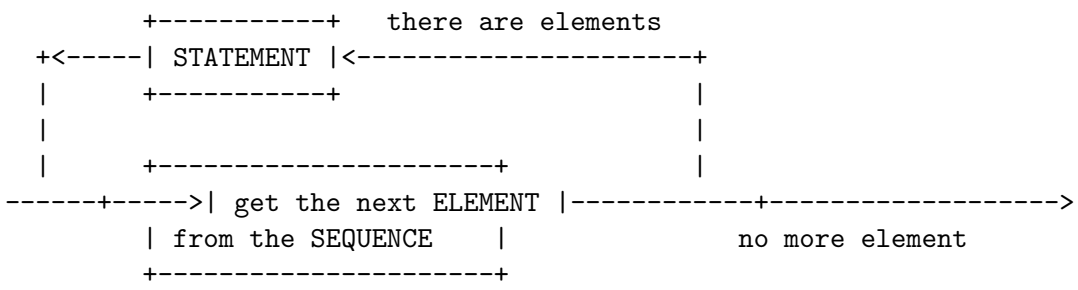
- Remarks:
- In a well written program, the condition eventually changes to false (otherwise there is an endless loop or empty loop).
- A while loop should be applied, if the number of iterations is not known at the beginning of the loop.

[21]:
```
# Example: The programming exam.
while int(input('How many points did you get? ')) < 16:
    print('You should learn more!')

print('Congratulations, you made it.')
```

```
How many points did you get? 12
You should learn more!
How many points did you get? 14
You should learn more!
How many points did you get? 20
Congratulations, you made it.
```

0.2.3 for statement

- Syntax: "' for ELEMENT in SEQUENCE: STATEMENT
- Flow diagram:



- Remarks:
- The sequence can be a consecutive sequence of integers, or another sequence (e.g. string, tuple, list, set, dict, open file).
- A for loop should be applied if A) the sequence is already available or B) we know the number of iterations at the beginning of the loop.

[22]:
```
# Creating a range.
r = range(0, 10)
r
```

[22]:
```
range(0, 10)
```

[23]:
```
list(r)
```

[23]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[24]:
```
# Example: Printing the first n square numbers.
n = int(input('n: '))
for i in range(1, n + 1):
    print(i**2)
```

```
n: 5
1
4
9
16
25
```

[28]:
```
# Example: Printing a * character n times.
n = int(input('n: '))
for i in range(n):
    print('*', end='')
```

```
n: 7
*******
```

[26]:
```
# ...shorter solution:
n = int(input('n: '))
print('*' * n)
```

```
n: 7
*******
```

[27]:
```
# Example: n-by-n triangle from * characters.
# *
# **
# ***
# ****

n = int(input('n: ')) # read n
for i in range(n):    # iterate over the rows
    print('*' * (i + 1))
```

```
n: 5
*
**
***
****
*****
```

[30]:
```
# Example: Counting vowels (in a lowercase text).
s = 'apple tree, pear tree'
```

```
vowels = {'a', 'e', 'i', 'o', 'u'}
nvowels = 0

for ch in s:
    if ch in vowels:
        nvowels += 1
print(nvowels)
```

```
8
```

[31]:
```
# Transform character to the next letter of the alphabet (for lowercase␣
↪letters).
# Letter 'z' should be transformed to 'a'.
ch = 'b'
idx = ord(ch) - ord('a')
chr((idx + 1) % 26 + ord('a'))
```

[31]:
```
'c'
```

[32]:
```
# Example: Caesar-encoding (for lowercase text without whitespaces)
s = 'venividivici' # the original text
t = ''             # the encoded text
offset = 3

for ch in s:
    idx = ord(ch) - ord('a')
    ch_encoded = chr((idx + offset) % 26 + ord('a'))
    t += ch_encoded

print(t)
```

```
yhqlylglylfl
```

0.3 Exercise: Simple number guessing game

Write a program that draws a random number from 1 to 100, then asks for guesses from the player, until the player finds out the number. After each guess, the program should show if the given guess was too low, too high or it was correct.

[34]:
```
# Importing the (pseudo-)random number generator module.
import random

# Drawing a random number from 1 to 100.
x = random.randint(1, 100)

# Asking for guesses until the player finds out the number.

```

```
y = None # the guess of the player
while y != x:
    # read guess
    y = int(input('Guess: '))

    # 3-way branching
    if y < x:
        print('Too low.')
    elif y > x:
        print('Too high.')
    else:
        print('CORRECT!')
```

```
Guess: 50
Too high.
Guess: 25
Too high.
Guess: 12
Too high.
Guess: 7
Too high.
Guess: 3
CORRECT!
```

04_en

May 20, 2022

0.1 Comprehensions

Comprehension is a language element that enables the concise definition of sequences. Comprehension is somewhat similar to set builder notation used in mathematics (example: the set of odd numbers can be given as $\{2k+1 \mid k \in \mathbb{Z}\}$).

0.1.1 Unconditional comprehension

[1]:
```
# Prepare the list of the first 10 square numbers using an accumulator variable!
l = []
for i in range(1, 11):
    l.append(i**2)
print(l)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[2]:
```
# The same with a list comprehension:
l = [i**2 for i in range(1, 11)]
print(l)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[3]:
```
# Prepare the set of the first 10 square numbers using an accumulator variable!
s = set()
for i in range(1, 11):
    s.add(i**2)
print(s)
```

```
{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
```

[4]:
```
# The same with a set comprehension:
s = {i**2 for i in range(1, 11)}
print(s)
```

```
{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
```

[5]:
```
# Prepare a dict that assigns their ASCII code to lowercase vowels!
# Use an accumulator variable!
d = {}
for c in 'aeiou':
    d[c] = ord(c)
print(d)
```

```
{'a': 97, 'e': 101, 'i': 105, 'o': 111, 'u': 117}
```

[6]:
```
# The same with a dict comprehension:
d = {c: ord(c) for c in 'aeiou'}
print(d)
```

```
{'a': 97, 'e': 101, 'i': 105, 'o': 111, 'u': 117}
```

[7]:
```
# Exercise: Switch pair members in a list.
pairs = [('apple', 10), ('pear', 20), ('peach', 30)]
[(p[1], p[0]) for p in pairs]
```

[7]:
```
[(10, 'apple'), (20, 'pear'), (30, 'peach')]
```

0.1.2 Conditional comprehension

[8]:
```
# Conditional list comprehension.
[i**2 for i in range(1, 11) if i % 2 == 0]
```

[8]:
```
[4, 16, 36, 64, 100]
```

[9]:
```
# Conditional set comprehension.
{i**2 for i in range(1, 11) if i % 2 == 0}
```

[9]:
```
{4, 16, 36, 64, 100}
```

[10]:
```
# Conditional dict comprehension.
{c: ord(c) for c in 'aeiou' if c != 'u'}
```

[10]:
```
{'a': 97, 'e': 101, 'i': 105, 'o': 111}
```

1 Statistical functions

[11]:
```
# Calculate the sum of numbers with sum function (where the data is an iterable␣
↪object)

# The data is a list contains integer numbers
```

```
print(sum([1, 8, 3]))

# The data is a list contains float numbers
print(sum([1.1, 2.2]))

# The data is a tuple contains integer and float numbers
print(sum((1, 2.3)))

# The data is a set with complex numbers)
print(sum({1 + 2j, 2 + 3j}))
```

```
12
3.3000000000000003
3.3
(3+5j)
```

[12]:
```
# If the elements are not numbers we get in error
sum([3, 'apple'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-97c6421847ae> in <module>
      1 # If the elements are not numbers we get in error
----> 2 sum([3, 'apple'])

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

[13]:
```
# Calculate the minimum/maximum value (where the data have to be comparable)

# The data are numbers
print(min(3, 5, 2.8))

# The data is a list contains numbers
print(max([3, 5, 2.8]))

# The data is a string (as a sequence of characters)
print(min('apple'))
print(max('apple'))

# The data are strings
print(min('Little John', 'Lady Mariann', 'Robin Hood'))

# The data is a list contains strings
print(max(['Little John', 'Lady Mariann', 'Robin Hood']))
```

```
2.8
5
```

```
a
p
Lady Mariann
Robin Hood
```

[14]:
```
# If the elements are not comparable we get an error
min(3, 'apple')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-88719b966d13> in <module>
      1 # If the elements are not comparable we get an error
----> 2 min(3, 'apple')

TypeError: '<' not supported between instances of 'str' and 'int'
```

1.1 Sorting

[15]:
```
# Sorting a list in place.
l = [10, 2, 11, 3]
l.sort()
```

[16]:
```
l
```

[16]:
```
[2, 3, 10, 11]
```

[17]:
```
# Sorting to descending order.
l = [10, 2, 11, 3]
l.sort(reverse=True)
```

[18]:
```
l
```

[18]:
```
[11, 10, 3, 2]
```

[19]:
```
# The list items have to be comparable!
l = [1, 2, 'apple']
l.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-25c58c018ee3> in <module>
      1 # The list items have to be comparable!
      2 l = [1, 2, 'apple']
----> 3 l.sort()
```

```
TypeError: '<' not supported between instances of 'str' and 'int'
```

[20]:
```
# If the list contains only strings, then it can be sorted.
l = ['apple', 'pear', 'cherry']
l.sort()
```

[21]:
```
l
```

[21]:
```
['apple', 'cherry', 'pear']
```

[22]:
```
# Sorting a collection into a list.
l1 = [10, 4, 20, 5]
l2 = sorted(l1)
```

[23]:
```
l2
```

[23]:
```
[4, 5, 10, 20]
```

[24]:
```
l1
```

[24]:
```
[10, 4, 20, 5]
```

[25]:
```
# A tuple can also be sorted into a new list.
sorted(('c', 'a', 'b'))
```

[25]:
```
['a', 'b', 'c']
```

[26]:
```
# ...and a set too.
sorted({'c', 'a', 'b'})
```

[26]:
```
['a', 'b', 'c']
```

[27]:
```
# For a dict, the keys will be sorted.
sorted({'b': 20, 'a': 20})
```

[27]:
```
['a', 'b']
```

[28]:
```
# Sorting a list of pairs (lexicographically).
l = [('beer', 10), ('wine', 20), ('rum', 30), ('wine', 5)]
sorted(l)
```

[28]:
```
[('beer', 10), ('rum', 30), ('wine', 5), ('wine', 20)]
```

Remark: Python contains a stable sorting algorithm.

1.2 File handling

- A file is a collection of data stored in one unit.
- The life cycle of a file consists looks like as follows:

1. opening
2. reading, writing, positioning, ...
3. closing

[29]:
```
# Opening a file.
f = open('example_file.txt')
```

We can create `example_file.txt` in the working directory using the menu item New / Text File.

[30]:
```
# Closing the file.
f.close()
```

[31]:
```
# Reading the content of the file into a string.
f = open('example_file.txt')
s = f.read()
f.close()
print(s)
```

```
# example data
apple,10
pear,20
cherry,30
```

[32]:
```
# ...shorter version of the same:
s = open('example_file.txt').read()
print(s)
```

```
# example data
apple,10
pear,20
cherry,30
```

[33]:
```
# Reading the first 2 lines.
f = open('example_file.txt')
s1 = f.readline()
s2 = f.readline()
f.close()
print(s1)
print(s2)
```

```
# example data
```

```
apple,10
```

[34]:
```
# Remark: readline puts the line break too into the result.
s1
```

[34]:
```
'# example data\n'
```

[35]:
```
# The line break can be removed using e.g. strip:
s1.strip()
```

[35]:
```
'# example data'
```

[36]:
```
# Reading the lines of a file into a string list.
open('example_file.txt').readlines()
```

[36]:
```
['# example data\n', 'apple,10\n', 'pear,20\n', 'cherry,30\n']
```

[37]:
```
# Splitting a string along a delimiter sequence (tokenization).
line = 'aa,bb,ccc'
line.split(',')
```

[37]:
```
['aa', 'bb', 'ccc']
```

[38]:
```
# By default, the delimiters are whitespaces and empty strings are removed from␣
 ↪the result.
line = 'aa  bb\tccc\n'
line.split()
```

[38]:
```
['aa', 'bb', 'ccc']
```

[39]:
```
# Iterating over the lines of a text file.
for line in open('example_file.txt'):
    print(line)
```

```
# example data

apple,10

pear,20

cherry,30
```

[40]:
```
# Skipping the first line of the file and tokenizing further lines.
data = []
f = open('example_file.txt')
f.readline() # skip the first line
```

```
for line in f:
    tok = line.strip().split(',')
    record = tok[0], int(tok[1])
    data.append(record)
f.close()
```

[41]:
```
data
```

[41]:
```
[('apple', 10), ('pear', 20), ('cherry', 30)]
```

[42]:
```
# Writing a string into file.
f = open('example_file_2.txt', 'w')
f.write('hello')
f.close()
```

[43]:
```
# ...a shorter version of the same:
open('example_file_2.txt', 'w').write('hello')
# (The CPython interpreter closes the file immediately, as there is no more␣
 ↪references to it.)
```

[43]:
```
3
```

[44]:
```
# Preparing a file that contains a Celsius-Fahrenheit table.
file = open('celsius_fahrenheit.txt', 'w')
file.write('Celsius\tFahrenheit\n')
for c in range(-20, 41, 5):
    f = c * 9 / 5 + 32
    file.write(f'{c}\t{f}\n')
file.close()
```

[45]:
```
# Determine the set of words contained in the text file real_programmers.txt!
{line.strip() for line in open('real_programmers.txt')}
```

[45]:
```
{'accounting',
 'all',
 'artificial',
 'at',
 'do',
 'fortran',
 'if',
 'in',
 'intelligence',
 'it',
 'list',
 'manipulation',
 'processing',
 'programmers',
```

```
 'programs',
 'real',
 'string',
 'they'}
```

[46]:
```
# Read the content of matrix.txt into a list of int lists!
matrix = []
for line in open('matrix.txt'):
    tok = line.strip().split()
    matrix.append([int(t) for t in tok])
matrix
```

[46]:
```
[[0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
 [0, 0, 1, 0, 1, 1, 0, 1, 0, 1],
 [0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 1, 1, 0, 0],
 [1, 0, 1, 1, 0, 0, 1, 0, 1, 1],
 [1, 0, 1, 0, 0, 1, 1, 0, 1, 0],
 [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
 [0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 0, 1, 0, 1, 1, 1, 0, 0],
 [1, 0, 1, 0, 1, 0, 0, 1, 0, 1]]
```

[47]:
```
# The same using a double comprehension:
matrix = [[int(t) for t in l.strip().split()] for l in open('matrix.txt')]
matrix
```

[47]:
```
[[0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
 [0, 0, 1, 0, 1, 1, 0, 1, 0, 1],
 [0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 1, 1, 0, 0],
 [1, 0, 1, 1, 0, 0, 1, 0, 1, 1],
 [1, 0, 1, 0, 0, 1, 1, 0, 1, 0],
 [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
 [0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 0, 1, 0, 1, 1, 1, 0, 0],
 [1, 0, 1, 0, 1, 0, 0, 1, 0, 1]]
```

1.3 Exercise: Word statistics in Hamlet

The file `hamlet.txt` contains the text of Hamlet. Write a program that computes and prints the 30 most frequent words in the text. The definition of word should be the following:

- Words are separated from each other by whitespaces (space, tab, line break).
- No distinction should be made between lower- and uppercase letters.
- Trailing and ending punctuation characters should be discarded.

[48]:
```
# Reading text, converting to lowercase, splitting.
words = open('hamlet.txt').read().lower().strip().split()
```

[49]:
```
# Remove punctuation from the beginning and end of words.
import string
words = [w.strip(string.punctuation) for w in words]
```

[50]:
```
# Compute word frequencies.
freq = {}
for w in words:
    if w in freq: freq[w] += 1
    else: freq[w] = 1
```

[51]:
```
# Convert dict to list of pairs.
freq2 = [(x[1], x[0]) for x in freq.items()]
```

[52]:
```
# Sorting.
freq2.sort(reverse=True)
```

[53]:
```
# Printing 30 most common words.
for i in range(30):
    print(freq2[i])
```

```
(1145, 'the')
(973, 'and')
(736, 'to')
(674, 'of')
(565, 'i')
(539, 'you')
(534, 'a')
(513, 'my')
(431, 'in')
(409, 'it')
(381, 'that')
(358, 'ham')
(339, 'is')
(310, 'not')
(297, 'this')
(297, 'his')
(268, 'with')
(258, 'but')
(248, 'for')
(241, 'your')
(231, 'me')
(223, 'lord')
(219, 'as')
(216, 'be')
```

```
(213, 'he')
(200, 'what')
(195, 'king')
(195, 'him')
(194, 'so')
(180, 'have')
```

05_en

May 20, 2022

0.1 Functions

0.1.1 Concepts

- A function is a named subprogram that can be called from other parts of the program.
- By using functions, computing tasks can be divided into smaller units. The code of frequently used functions can be organized into libraries.
- In mathematics, a function has no side effects. In Python they can have!
- In Python, functions are ”first class citizens”:
- A function can be assigned to a variable as a value.
- Functions can be nested into each other.
- A function can get a function as a parameter and it can return a function as the result.
- It is important to differentiate between *function definition* and *function call*:
- A function's definition specifies what output is assigned to what input (and what side effects are executed). A function's definition usually appears in the program only once (otherwise the last definition will be the valid one).
- Calling a function means that we compute the assigned value for a given input. A defined function can be called in a program multiple times.

[12]:
```
# Example: Definition of the n-th root function.
def root(x, n=2):
    '''Returns the n-th root of x.'''
    return x**(1 / n)
```

If the first statement of a function is a string, then this will be the documentation string (docstring).

[13]:
```
# Querying the docstring.
root.__doc__ # 'dunder' doc
```

[13]:
```
'Returns the n-th root of x.'
```

[14]:
```
# The __doc__ attribute is an ordinary string, we can use it in arbitrary␣
 ↪operations.
root.__doc__ *= 2
```

- In Python, a function can have *positional* and *keyword* arguments.
- In the function definition, first the positional, then the keyword arguments are enlisted.
- Positional arguments have no default value, keyword arguments do have.
- A function can have zero positional and/or keyword arguments.
- At a function call...
- The value of all positional arguments have to be specified, in the order given in the definition.
- Specifying the value of the keyword arguments is not mandatory.

[4]:
```
# Computing the square root of 2.
root(2)
```

[4]:
```
1.4142135623730951
```

[5]:
```
root(2, n=2)
```

[5]:
```
1.4142135623730951
```

[6]:
```
root(2, 2)
```

[6]:
```
1.4142135623730951
```

[5]:
```
# Computing the cube root of 2.
root(2, n=3)
```

[5]:
```
1.2599210498948732
```

[6]:
```
# The second argument does not have to be named.
root(2, 3)
```

[6]:
```
1.2599210498948732
```

[9]:
```
# A variable can get a function as a value.
f = root
```

[10]:
```
type(root)
```

[10]:
```
function
```

[11]:
```
type(f)
```

[11]:
```
function
```

[12]:
```
f(2)
```

[12]:
```
1.4142135623730951
```

[7]:
```
# Dummy example for nesting and returning a function.
def f(y):
    def g(x):
        return x * y
    return g
```

[8]:
```
g2 = f(2) # Doubling function.
g3 = f(3) # Tripling function.
```

[15]:
```
g2(10)
```

[15]: 20

[16]:
```
g3(20)
```

[16]: 60

0.1.2 Exercises

Prime testing Write a function that decides if a natural number is prime or not!

[9]:
```
# Version 1: without a function

n = 7
p = True
for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
        p = False
        break # exits from the actual for or while loop
if n == 1:
    p = False
print(p)
```

```
True
```

[15]:
```
# Version 2: with a function

def is_prime(n):
    '''Returns True if n is prime, returns False otherwise.'''
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return n != 1
```

[16]:
```
is_prime(80)
```

[16]: False

```
[17]: is_prime(17)
```

[17]: True

Greatest common divisor Write a function for determining the greatest common divisor of two natural numbers!

```
[18]: # Version 1: without a function

a = 20
b = 12
gcd = 1
for i in range(1, min(a, b) + 1):
    if (a % i == 0) and (b % i == 0):
        gcd = i
print(gcd)
```

4

```
[19]: # Version 2: with a function

def compute_gcd(a, b):
    gcd = 1
    for i in range(1, min(a, b) + 1):
        if (a % i == 0) and (b % i == 0):
            gcd = i
    return gcd
```

```
[20]: compute_gcd(2, 14)
```

[20]: 2

```
[21]: compute_gcd(6, 18)
```

[21]: 6

Quadratic equation solver Write a function for solving quadratic equations!

```
[22]: def solve_quadratic(a, b, c):
    '''Solve the equation a*x^2 + b*x + c = 0
    and return the list of solutions.'''

    # compute d
    d = b**2 - 4 * a * c
```

```
    if d > 0: # 2 solutions
        return [(-b + d**0.5) / (2 * a),
                (-b - d**0.5) / (2 * a)]
    elif d == 0: # 1 solution
        return [-b / (2 * a)]
    else:
        return []
```

[23]:
```
solve_quadratic(1, 3, 2)
```

[23]:
```
[-1.0, -2.0]
```

[24]:
```
solve_quadratic(1, 2, 1)
```

[24]:
```
[-1.0]
```

[25]:
```
solve_quadratic(1, 1, 10)
```

[25]:
```
[]
```

0.2 Lambda expressions

- A lambda expression in Python is one-liner, anonymous function.
- (Other elements of functional programming in Python: map, filter.)

[26]:
```
# Example lambda expression.
f = lambda x: x + 42
f(3)
```

[26]:
```
45
```

[27]:
```
# More than one inputs are also allowed.
g = lambda x, y: x + y
g(22, 33)
```

[27]:
```
55
```

[28]:
```
# ...or no input.
h = lambda: 33
h()
```

[28]:
```
33
```

0.2.1 Using lambda in sorting

[1]:
```
# Sorting a list of pairs by the second elements.
pairs = [('apple', 22), ('pear', 11), ('peach', 33)]
sorted(pairs, key=lambda p: p[1])
```

[1]:
```
[('pear', 11), ('apple', 22), ('peach', 33)]
```

[2]:
```
# The solution without a lambda expression.
pairs = [('apple', 22), ('pear', 11), ('peach', 33)]
def key_func(p):
    return p[1]
sorted(pairs, key=key_func)
```

[2]:
```
[('pear', 11), ('apple', 22), ('peach', 33)]
```

[3]:
```
# Sorting dictionary keys by the assigned values.
words = {'king': 203, 'denmark': 24, 'queen': 192}
sorted(words, key=lambda w: words[w])
```

[3]:
```
['denmark', 'queen', 'king']
```

0.3 Exercise: Premier League standings

The file pl.txt contains the game results of Premier League 2011-12. Write a program that... - prints the percentage of games with at least one goal, - prints the game with the highest number of goals, - reads the value of n from the user and prints the standings after n rounds (sorting criteria: points, goal difference, goals scored).

[4]:
```
# Reading the data from file to list of dicts.

f = open('pl.txt')

# skip first 6 lines
for i in range(6):
    f.readline()

# process further lines
games = []
for line in f:
    tok = line.split('\t')
    game = {
        'round': int(tok[0]),
        'hteam': tok[1],
        'ateam': tok[2],
        'hgoals': int(tok[3]),
```

```
        'agoals': int(tok[4])
    }
    games.append(game)

f.close()
```

[5]:
```
len(games)
```

[5]: 380

[6]:
```
# Percentage of games with at least one goal.
count = 0
for g in games:
    if g['hgoals'] + g['agoals'] > 0:
        count += 1
print(count / len(games) * 100)
```

92.89473684210526

[7]:
```
# ...the same in one line:
sum([g['hgoals'] + g['agoals'] > 0 for g in games]) / len(games) * 100
```

[7]: 92.89473684210526

[9]:
```
# The game with the highest number of goals.
goals_max = 0
for g in games:
    goals = g['hgoals'] + g['agoals']
    if goals > goals_max:
        goals_max = goals
        game_max = g

print(goals_max, game_max)
```

10 {'round': 3, 'hteam': 'Manchester United', 'ateam': 'Arsenal FC', 'hgoals': 8, 'agoals': 2}

[10]:
```
# ...the same in one line:
max(games, key=lambda x: x['hgoals'] + x['agoals'])
```

[10]:
```
{'round': 3,
 'hteam': 'Manchester United',
 'ateam': 'Arsenal FC',
 'hgoals': 8,
 'agoals': 2}
```

[11]:
```
# Standings after n rounds.

def update_stats(hteam, ateam, hgoals, agoals, stats):
    # update points
    if hgoals > agoals: stats[hteam][0] += 3
    elif hgoals == agoals: stats[hteam][0] += 1

    # update goal difference
    stats[hteam][1] += hgoals - agoals

    # update goals scored
    stats[hteam][2] += hgoals

# read n
n = int(input('n: '))

# initialize team statistics (points, goal difference, goals scored)
stats = {g['hteam']: [0, 0, 0] for g in games}

# compute statistics
for g in games:
    if g['round'] <= n:
        update_stats(g['hteam'], g['ateam'], g['hgoals'], g['agoals'], stats)
        update_stats(g['ateam'], g['hteam'], g['agoals'], g['hgoals'], stats)
```

```
n: 2
```

[12]:
```
# sorting
standings = sorted(stats.items(), key=lambda p: tuple(p[1]), reverse=True)
```

[13]:
```
# printing
for s in standings:
    team = s[0]
    pts = s[1][0]
    gd =  s[1][1]
    gs = s[1][2]
    print(f'{team:25}{gd:5}{gs:5}{pts:5}')
```

```
Manchester City             5    7    6
Manchester United           4    5    6
Wolverhampton Wanderers     3    4    6
Liverpool FC                2    3    4
Aston Villa                 2    3    4
Chelsea FC                  1    2    4
Newcastle United            1    1    4
Bolton Wanderers            3    6    3
Tottenham Hotspur          -1    2    3
Queens Park Rangers        -3    1    3
```

```
Norwich City                0    2    2
Wigan Athletic              0    1    2
Stoke City                  0    1    2
Sunderland AFC             -1    1    1
Fulham FC                  -2    0    1
Arsenal FC                 -2    0    1
Swansea City               -4    0    1
West Bromwich Albion       -2    2    0
Blackburn Rovers           -3    2    0
Everton FC                 -3    0    0
```

[15]:
```python
# ...shorter solution:
for team, (pts, gd, gs) in standings:
    print(f'{team:25}{gd:5}{gs:5}{pts:5}')
```

```
Manchester City             5    7    6
Manchester United           4    5    6
Wolverhampton Wanderers     3    4    6
Liverpool FC                2    3    4
Aston Villa                 2    3    4
Chelsea FC                  1    2    4
Newcastle United            1    1    4
Bolton Wanderers            3    6    3
Tottenham Hotspur          -1    2    3
Queens Park Rangers        -3    1    3
Norwich City                0    2    2
Wigan Athletic              0    1    2
Stoke City                  0    1    2
Sunderland AFC             -1    1    1
Fulham FC                  -2    0    1
Arsenal FC                 -2    0    1
Swansea City               -4    0    1
West Bromwich Albion       -2    2    0
Blackburn Rovers           -3    2    0
Everton FC                 -3    0    0
```

06_en

May 20, 2022

0.1 Unpacking

[3]:
```
# General case.
x, y = [10, [20, (2, 3, 4)]]
```

[4]:
```
print(x)
print(y)
```

```
10
[20, (2, 3, 4)]
```

[5]:
```
x, (y, z) = [10, [20, (2, 3, 4)]]
```

[6]:
```
print(x)
print(y)
print(z)
```

```
10
20
(2, 3, 4)
```

[7]:
```
# If the two sides of the assignment are not compatible, we get an error.
x, y, z = [10, [20, (2, 3, 4)]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-a8a7679b6a6c> in <module>
      1 # If the two sides of the assignment are not compatible, we get an error.
----> 2 x, y, z = [10, [20, (2, 3, 4)]]

ValueError: not enough values to unpack (expected 3, got 2)
```

[11]:
```
# Multiple assignment.
a, b = 10, 20
print(a)
print(b)
```

```
10
20
```

[13]:
```
# Swap values.
x = 100
y = 200
x, y = y, x
print(x)
print(y)
```

```
200
100
```

[14]:
```
# Unpacking applied in a for loop.
data = [('apple', 10), ('cherry', 20)]
for x, y in data:
    print(x)
    print(y)
```

```
apple
10
cherry
20
```

0.2 Slicing

- The syntax of the slice notation is [lower bound: upper bound: step size].
- The selection interval is open from above, meaning that the upper bound specifies the first index that is not selected.

[15]:
```
# Examples of slicing.
data = [1, 2, 'apple', 'pear', 1.5]
```

[16]:
```
data[1:3:1]
```

[16]:
```
[2, 'apple']
```

[17]:
```
data[1:3] # the step size can be omitted
```

[17]:
```
[2, 'apple']
```

[18]:
```
# The lower and upper bound and also the step size can be omitted.
data[:4]
```

[18]:
```
[1, 2, 'apple', 'pear']
```

[19]:
```
data[3:]
```

[19]:
```
['pear', 1.5]
```

[20]:
```
data[1::2]
```

[20]:
```
[2, 'pear']
```

[21]:
```
# We can use negative indices, minus 1 corresponds to the last item.
data[-1]
```

[21]:
```
1.5
```

[22]:
```
data[-2]
```

[22]:
```
'pear'
```

[23]:
```
data[:-1] # all elements except the last
```

[23]:
```
[1, 2, 'apple', 'pear']
```

[24]:
```
data[-3:] # last 3 elements
```

[24]:
```
['apple', 'pear', 1.5]
```

0.3 Advanced iteration techniques

0.3.1 enumerate

[23]:
```
# Iterating over both the items and indices, ordinary solution.
x = ['apple', 'cherry', 'orange']
for i in range(len(x)):
    print(i, x[i])
```

```
0 apple
1 cherry
2 orange
```

[22]:
```
# A more elegant solution, using enumerate:
x = ['apple', 'cherry', 'orange']
for i, xi in enumerate(x):
    print(i, xi)
```

```
0 apple
1 cherry
2 orange
```

[24]:
```
# The result of enumerate is an iterable object.
type(enumerate(x))
```

[24]:
```
enumerate
```

[26]:
```
# Conversion to a list of pairs.
list(enumerate(x))
```

[26]:
```
[(0, 'apple'), (1, 'cherry'), (2, 'orange')]
```

[27]:
```
# Using enumerate without unpacking.
for a in enumerate(x):
    print(a[0], a[1])
```

```
0 apple
1 cherry
2 orange
```

[31]:
```
# Using enumerate to account line numbers at file processing.
for i, line in enumerate(open('example_file.txt')):
    print(i, line)
```

```
0 # example data

1 apple,10

2 pear,20

3 cherry,30
```

0.3.2 zip

[34]:
```
# Iterating over multiple sequences in parallel, ordinary solution.
x = ['apple', 'cherry', 'orange']
y = [100, 200, 300]
for i in range(len(x)):
    print(x[i], y[i])
```

```
apple 100
cherry 200
orange 300
```

[37]:
```
# A more elegant solution, using zip:
x = ['apple', 'cherry', 'orange']
y = [100, 200, 300]
for xi, yi in zip(x, y):
```

```
    print(xi, yi)
```

```
apple 100
cherry 200
orange 300
```

[38]:
```
# The result of zip is an iterable object.
type(zip(x, y))
```

[38]:
```
zip
```

[39]:
```
# Conversion to a list.
list(zip(x, y))
```

[39]:
```
[('apple', 100), ('cherry', 200), ('orange', 300)]
```

[40]:
```
# Using zip without unpacking.
for a in zip(x, y):
    print(a[0], a[1])
```

```
apple 100
cherry 200
orange 300
```

[41]:
```
# If the sequences have different sizes, then the result will get the length of␣
 ↪the shorter one.
x = ['apple', 'cherry', 'orange', 'banana']
y = [100, 200, 300]
for xi, yi in zip(x, y):
    print(xi, yi)
```

```
apple 100
cherry 200
orange 300
```

[42]:
```
# Also, zip can be applied to more than two sequences.
x = ['apple', 'cherry', 'orange']
y = [100, 200, 300]
z = [1.5, 2.5, 3.5]
for xi, yi, zi in zip(x, y, z):
    print(xi, yi, zi)
```

```
apple 100 1.5
cherry 200 2.5
orange 300 3.5
```

0.4 Exercises / 1

[58]:
```
# Write a program that creates a dict from a string
# of the following format: 'KEY_1,VALUE_1,...KEY_n,VALUE_n'.

s = 'apple,10,pear,20,banana,30,orange,40'
t = s.split(',')
d = dict(zip(t[::2], [int(x) for x in t[1::2]]))
d
```

[58]:
```
{'apple': 10, 'pear': 20, 'banana': 30, 'orange': 40}
```

[64]:
```
# Write a program that creates a difference sequence from the number sequence␣
↪[a_1, ..., a_n],
# so that i-th element of the difference sequence is a_{i+1} - a_i.

a = [10, 12, 15, 16, 22, 33, 40]
```

[65]:
```
# solution 1
diff = [a[i + 1] - a[i] for i in range(len(a) - 1)]
diff
```

[65]:
```
[2, 3, 1, 6, 11, 7]
```

[67]:
```
i = 0
diff = []
while i < len(a) - 1:
    diff.append(a[i + 1] - a[i])
    i += 1
print(diff)
```

```
[2, 3, 1, 6, 11, 7]
```

[74]:
```
diff = [x - y for x, y in zip(a[1:], a[:-1])]
diff
```

[74]:
```
[2, 3, 1, 6, 11, 7]
```

[ ]:
```
# Write a program that simulates a 5-from-90 lottery draw,
# without using the function random.sample.
```

[79]:
```
# solution 1 ("ugly")
import random
res = set()
while len(res) < 5:
    res.add(random.randint(1, 90))
print(res)
```

```
{73, 44, 60, 30, 31}
```

[83]:
```
# solution 2 ("nice")
pool = list(range(1, 91))
for i in range(5):
    j = random.randint(i, 89)
    pool[i], pool[j] = pool[j], pool[i]
print(pool[:5])
```

```
[22, 33, 5, 20, 89]
```

[86]:
```
# ...using random.sample:
random.sample(range(1, 91), 5)
```

[86]:
```
[6, 70, 58, 55, 61]
```

0.5 Exercises / 2

Write a program that converts an arabic number between 1 and 99 to a roman numeral!

[12]:
```
# tens ones
#1        X        I
#2        XX        II
#3        XXX        III
#4        XL        IV
#5        L        V
#6        LX        VI
#7        LXX        VII
#8        LXXX        VIII
#9        XC        IX

n = 47 # XL VII
o = n % 10
t = n // 10
roman_ones = ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX']
roman_tens = ['', 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX', 'LXXX', 'XC']
roman_tens[t] + roman_ones[o]
```

[12]:
```
'XLVII'
```

[13]:
```
def arabic_to_roman(n):
    o = n % 10
    t = n // 10
    roman_ones = ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX']
    roman_tens = ['', 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX', 'LXXX', 'XC']
    return roman_tens[t] + roman_ones[o]
```

[14]:
```
arabic_to_roman(47)
```

[14]: 'XLVII'

[15]:
```
arabic_to_roman(8)
```

[15]: 'VIII'

[16]:
```
arabic_to_roman(93)
```

[16]: 'XCIII'

Write a program that converts a roman numeral between I and XCIX to an arabic number!

[19]:
```
r2a = {arabic_to_roman(i): i for i in range(1, 100)}
```

[21]:
```
r2a['XIII']
```

[21]: 13

[22]:
```
r2a['LXXXIV']
```

[22]: 84

07_en

May 20, 2022

0.1 Modules and packages

Module: A file with extension .py. - It contains definitions and statements. - If the module is implemented in the file `xyz.py`, then the module can be referenced as `xyz`. - Modules can be imported from other Python programs.

Package: Collection of modules. - A package can contain subpackages/submodules. The hierarchy is defined by the directory structure of the package. - Standard packages and modules form the standard library and do not require installation. - The official repository of external packages is PyPI (https://pypi.python.org/pypi).

[4]:
```
# Importing a module/package.
import random
```

[5]:
```
random.randint(1, 10)
```

[5]: 7

[6]:
```
# Importing a function from a module/package.
from random import randint
```

[7]:
```
randint(2, 20)
```

[7]: 11

[8]:
```
# Import the full content of a module/package. (This solution should be avoided␣
↪in most cases.)
from random import *
```

[9]:
```
randrange(10)
```

[9]: 0

[10]:
```
# Importing a function from a submodule/subpackage.
from os.path import dirname
```

[11]:
```
dirname('/tmp/foo/a.txt')
```

[11]:
```
'/tmp/foo'
```

[12]:
```
# Importing a module/package using a shorter name.
import random as rnd
rnd.randint(1, 10)
```

[12]:
```
7
```

0.2 Elements of the standard library I.

- Python's standard library contains more than 200 packages and modules. It provides standard solutions to everyday programming tasks.
- In the course we only attempt to overview a small subset of the standard library.
- A good programmer does not invent the wheel. If possible, he/she solves the problem with the tools of the standard library.

datetime

- Provides tools for date and time handling.
- Supports date arithmetic, handles time zones, daylight time, leap years etc.
- Allows dates both with and without time zone.

[1]:
```
import datetime
```

[2]:
```
# Defining time with microsecond accuracy.
datetime.datetime(2020, 10, 12, 14, 0, 25, 100)
```

[2]:
```
datetime.datetime(2020, 10, 12, 14, 0, 25, 100)
```

[3]:
```
# Defining a date with day accuracy.
datetime.date(2000, 9, 21)
```

[3]:
```
datetime.date(2000, 9, 21)
```

[4]:
```
# Time arithmetic.
dt1 = datetime.datetime(2020, 10, 12, 14, 0, 25, 100)
dt2 = datetime.datetime(2018, 2, 11, 0, 0, 0)
diff = dt1 - dt2
diff
```

[4]:
```
datetime.timedelta(days=974, seconds=50425, microseconds=100)
```

[5]:
```
# The difference is 974 days + 50425 seconds + 100 microseconds.
print(diff.days)
print(diff.seconds)
print(diff.microseconds)
```

```
974
50425
100
```

[6]:
```
# Given in seconds:
diff.total_seconds()
```

[6]:
```
84204025.0001
```

[7]:
```
# Add 8 hours to the time!
dt = datetime.datetime(2020, 10, 15, 20, 5, 5)
dt + datetime.timedelta(0, 3600 * 8)
```

[7]:
```
datetime.datetime(2020, 10, 16, 4, 5, 5)
```

[8]:
```
# Querying the current time.
dt = datetime.datetime.now()
dt
```

[8]:
```
datetime.datetime(2022, 2, 6, 14, 17, 30, 935848)
```

[9]:
```
# Accessing the fields of the datetime object.
print(dt.year)
print(dt.month)
print(dt.day)
print(dt.hour)
print(dt.minute)
print(dt.second)
print(dt.microsecond)
```

```
2022
2
6
14
17
30
935848
```

[10]:
```
# Querying the day of week (0=Monday, ..., 6=Sunday).
dt.weekday()
```

[10]:
```
6
```

[11]:
```
# Exercise: Write a program that reads a date, then prints the day number
# of the date within the given year (how manyth is the day within the year)!

# read date
tok = input('Enter a date (Y-M-D): ').split('-')
```

```
y, m, d = int(tok[0]), int(tok[1]), int(tok[2])

# day number
dt1 = datetime.datetime(y, m, d)
dt2 = datetime.datetime(y, 1, 1)
print((dt1 - dt2).days + 1)
```

```
Enter a date (Y-M-D): 2000-12-31
366
```

[12]:
```
# Exercise: Write a program that prints the names
# of the following people, ordered by ascending age.

people = [
    # name, date of birth
    ('Gipsz Jakab', datetime.date(1957, 11, 21)),
    ('Wincs Eszter', datetime.date(1980, 5, 7)),
    ('Békés Farkas', datetime.date(2014, 7, 30)),
    ('Har Mónika', datetime.date(1995, 2, 27)),
    ('Trab Antal', datetime.date(1961, 4, 1)),
    ('Git Áron', datetime.date(1995, 2, 28)),
    ('Bank Aranka', datetime.date(1980, 9, 1))
]
```

[13]:
```
sorted(people, key=lambda x: x[1], reverse=True)
```

[13]:
```
[('Békés Farkas', datetime.date(2014, 7, 30)),
 ('Git Áron', datetime.date(1995, 2, 28)),
 ('Har Mónika', datetime.date(1995, 2, 27)),
 ('Bank Aranka', datetime.date(1980, 9, 1)),
 ('Wincs Eszter', datetime.date(1980, 5, 7)),
 ('Trab Antal', datetime.date(1961, 4, 1)),
 ('Gipsz Jakab', datetime.date(1957, 11, 21))]
```

[14]:
```
# Exercise: Count the number of Friday the 13ths in the 20th century (from Jan 1, 1901 to Dec 31, 2000)!

# solution 1
s = 0
dt = datetime.date(1901, 1, 1)
while dt <= datetime.date(2000, 12, 31):
    if dt.day == 13 and dt.weekday() == 4:
        s += 1
    dt += datetime.timedelta(1)
print(s)
```

```
171
```

[15]:
```
# solution 2
s = 0
for y in range(1901, 2001):
    for m in range(1, 13):
        if datetime.date(y, m, 13).weekday() == 4:
            s += 1
print(s)
```

```
171
```

time

- Provides tools for low level time handling, such as measuring durations and waiting.

[1]:
```
import time
```

[2]:
```
# Querying the current time (as a UNIX time stamp).
time.time()
```

[2]:
```
1644153525.6518347
```

[4]:
```
import datetime

# The current date and time (calculating from the Unix time stamp)
print(datetime.datetime(1970,1,1) + datetime.timedelta(0,time.time()))
```

```
2022-02-06 13:19:33.333638
```

[5]:
```
# Measuring the duration of a calculation.
t0 = time.time()
s = 0
for i in range(1, 1000001):
    s += 1 / i**2
print(time.time() - t0)
print(s)
```

```
0.6059346199035645
1.64493306684877
```

[6]:
```
# Waiting for 2 seconds.
time.sleep(2)
```

math

- Contains basic mathematical functions.
- Advice: Do not use the math module in a NumPy based code, but use NumPy's built in functions instead!

[1]:
```
import math
```

[2]:
```
# Exponential function.
math.exp(1)
```

[2]: 2.718281828459045

[3]:
```
math.exp(2)
```

[3]: 7.38905609893065

[5]:
```
# Natural logarithm.
math.log(3)
```

[5]: 1.0986122886681098

[7]:
```
# Logarithm with base q.
math.log(8, 2)
```

[7]: 3.0

[8]:
```
# Trigonometric functions and their inverses.
math.sin(0)
```

[8]: 0.0

[9]:
```
math.cos(0)
```

[9]: 1.0

[12]:
```
math.sin(math.radians(45))
```

[12]: 0.7071067811865475

[13]:
```
# pi, e
math.pi
```

[13]: 3.141592653589793

[14]:
```
math.e
```

[14]: 2.718281828459045

[15]:
```
# Exercise: Write a program that reads a 2-dimensional vector from the user,
# and prints the angle between the vector and the (positive side of the) x axis!

vx = float(input('vx: '))
```

```
vy = float(input('vy: '))
```

```
vx: 1
vy: 1
```

[16]:
```
def compute_angle(vx, vy):
    if vx == 0:
        phi = math.pi / 2
    else:
        alpha = math.atan(abs(vy / vx))
        if vx > 0: phi = alpha
        else: phi = math.pi - alpha

    if vy < 0:
        phi *= -1
    return phi
```

[17]:
```
math.degrees(compute_angle(vx, vy))
```

[17]:
```
45.0
```

[18]:
```
math.degrees(compute_angle(-1, 1))
```

[18]:
```
135.0
```

[19]:
```
math.degrees(compute_angle(-1, -1))
```

[19]:
```
-135.0
```

[20]:
```
def compute_angle_v2(vx, vy):
    return math.atan2(vy, vx)
```

[22]:
```
print(math.degrees(compute_angle_v2(1, 1)))
print(math.degrees(compute_angle_v2(-1, 1)))
print(math.degrees(compute_angle_v2(-1, -1)))
```

```
45.0
135.0
-135.0
```

random

- Provides tools for generating pseudo-random numbers.

[24]:
```
import random
```

[33]:
```
# Drawing an int from a given interval.
random.randint(1, 100)
```

[33]:
```
100
```

[35]:
```
random.randrange(1, 101)
```

[35]:
```
72
```

[39]:
```
# Drawing a float from a given interval.
random.uniform(5, 10)
```

[39]:
```
7.328793385777218
```

[52]:
```
# Drawing from standard normal distribution.
random.gauss(0, 1)
```

[52]:
```
-0.1232751803020387
```

[56]:
```
# Setting the state of the random number generator.
random.seed(42)
print(random.gauss(0, 1))
print(random.gauss(0, 1))

random.seed(42)
print(random.gauss(0, 1))
print(random.gauss(0, 1))
```

```
-0.14409032957792836
-0.1729036003315193
-0.14409032957792836
-0.1729036003315193
```

[57]:
```
# Creating a random number generator object.
r1 = random.Random(42)
r2 = random.Random(42)
print(r1.uniform(5, 10))
print(r2.uniform(5, 10))
```

```
8.19713399228942
8.19713399228942
```

[25]:
```
# Drawing an item from a sequence.
seq = ['apple', 'pear', 'plum']
random.choice(seq)
```

[25]:
```
'apple'
```

[62]:
```
# Sampling without replacement.
random.sample(range(1, 91), 5)
```

[62]:
```
[76, 55, 5, 4, 12]
```

[63]:
```
sorted(random.sample(range(1, 91), 5))
```

[63]:
```
[4, 28, 30, 65, 78]
```

[71]:
```
# Exercise: Write a program that simulates a sequence of n coin tosses,
# then prints the number of heads and tails!
n = 100
data = [random.choice('HT') for i in range(n)]
print(data)
print(data.count('H'))
print(data.count('T'))
```

```
['T', 'H', 'H', 'H', 'T', 'H', 'H', 'H', 'H', 'T', 'T', 'T', 'H', 'H', 'T', 'H',
'T', 'H', 'T', 'H', 'H', 'T', 'H', 'H', 'T', 'H', 'T', 'H', 'H', 'T', 'T', 'T',
'T', 'T', 'H', 'H', 'T', 'T', 'H', 'H', 'T', 'T', 'H', 'H', 'H', 'H', 'T', 'H',
'H', 'H', 'T', 'H', 'T', 'T', 'T', 'T', 'T', 'T', 'H', 'H', 'H', 'H', 'T', 'H',
'H', 'H', 'H', 'H', 'H', 'H', 'T', 'T', 'T', 'T', 'H', 'H', 'T', 'T', 'T', 'H',
'H', 'T', 'H', 'T', 'H', 'H', 'H', 'T', 'H', 'H', 'H', 'H', 'T', 'T', 'T', 'H',
'T', 'T', 'T', 'T']
54
46
```

[89]:
```
# Exercise: Write a program that simulates a sequence of n coin tosses,
# then prints the length of the longest heads and tails sequence!

n = 20
data = [random.choice('HT') for i in range(n)]
print(' '.join(data))

def calc_longest_seq(data, symbol):
    x_prev = None
    actlen = 0
    maxlen = 0
    for x in data:
        if x == symbol:
            if x_prev == x: actlen += 1
            else: actlen = 1

            if actlen > maxlen:
                maxlen = actlen
        x_prev = x
    return maxlen
```

```
nh = calc_longest_seq(data, 'H')
nt = calc_longest_seq(data, 'T')

print(f'length of longest heads sequence: {nh}')
print(f'length of longest tails sequence: {nt}')
```

```
H H H T T T T T T H H H T T H T T T T T
length of longest heads sequence: 3
length of longest tails sequence: 6
```

08_en

May 20, 2022

0.1 Elements of the standard library II.

collections

- Provides specialized container data types.

[1]:
```
import collections
```

[3]:
```
# Dictionary for counting hashable objects.
data = 'abracadabra apple xyz'
c = collections.Counter(data)
print(c)
print(c['a'])
```

```
Counter({'a': 6, 'b': 2, 'r': 2, ' ': 2, 'p': 2, 'c': 1, 'd': 1, 'l': 1, 'e': 1,
'x': 1, 'y': 1, 'z': 1})
6
```

[8]:
```
# Word frequencies in Hamlet, computed with a Counter.

import string
text = open('hamlet.txt').read()
words = text.lower().split()
words = [w.strip(string.punctuation) for w in words]
freq = collections.Counter(words)
freq.most_common(30)
```

[8]:
```
[('the', 1145),
 ('and', 973),
 ('to', 736),
 ('of', 674),
 ('i', 565),
 ('you', 539),
 ('a', 534),
 ('my', 513),
 ('in', 431),
 ('it', 409),
 ('that', 381),
 ('ham', 358),
```

```
 ('is', 339),
 ('not', 310),
 ('his', 297),
 ('this', 297),
 ('with', 268),
 ('but', 258),
 ('for', 248),
 ('your', 241),
 ('me', 231),
 ('lord', 223),
 ('as', 219),
 ('be', 216),
 ('he', 213),
 ('what', 200),
 ('king', 195),
 ('him', 195),
 ('so', 194),
 ('have', 180)]
```

[12]:
```
# Dictionary with default values.
dd = collections.defaultdict(list)
```

[13]:
```
# Adding a new key-value pair.
dd['a'] = 10
```

[14]:
```
dd['a']
```

[14]:
```
10
```

[16]:
```
# Acessing a non-existent key.
# This will not raise an error, but assigns the default value to the key.
# The default value is created by the function call list().
dd['b']
```

[16]:
```
[]
```

[17]:
```
dd
```

[17]:
```
defaultdict(list, {'a': 10, 'b': []})
```

[18]:
```
# Accessing a non-existent key, then appending an item.
dd['c'].append(42)
```

[19]:
```
dd
```

[19]:
```
defaultdict(list, {'a': 10, 'b': [], 'c': [42]})
```

[21]:
```
# Convertsion to an ordinary dict.
d = dict(dd)
d
```

[21]:
```
{'a': 10, 'b': [], 'c': [42]}
```

[23]:
```
# Tuple with named items.
Game = collections.namedtuple('Game', ['round', 'hteam', 'ateam', 'hgoals',␣
↪'agoals'])
```

[26]:
```
g1 = Game(1, 'Arsenal FC', 'Chelsea', 2, 1)
g1
```

[26]:
```
Game(round=1, hteam='Arsenal FC', ateam='Chelsea', hgoals=2, agoals=1)
```

[28]:
```
Game(round=1, hteam='Arsenal FC', ateam='Chelsea', hgoals=2, agoals=1)
```

[28]:
```
Game(round=1, hteam='Arsenal FC', ateam='Chelsea', hgoals=2, agoals=1)
```

[30]:
```
# Tuple style usage.
print(g1[0])
print(g1[1])
```

```
1
Arsenal FC
```

[31]:
```
# Struct style usage.
print(g1.round)
print(g1.hteam)
```

```
1
Arsenal FC
```

[32]:
```
# namedtuple objects can be used as a dictionary keys.
{g1: 'apple'}
```

[32]:
```
{Game(round=1, hteam='Arsenal FC', ateam='Chelsea', hgoals=2, agoals=1):
 'apple'}
```

[9]:
```
# Exercise: Write a program that simulates n rolls with 2 dices,
# then prints how many times the sum of rolls was 2, 3, ..., 12!

from collections import Counter
from random import randint

n = 10000
```

```
# simulating n rolls with 2 dices
rolls = [randint(1, 6) + randint(1, 6) for i in range(n)]

# compute & display frequencies
freq = Counter(rolls)
for s in range(2, 13):
    print(s, freq[s])
```

```
2 278
3 557
4 842
5 1079
6 1413
7 1719
8 1362
9 1116
10 828
11 546
12 260
```

copy

- Contains a shallow and a deep copy function.

[10]:
```
import copy
```

[12]:
```
# In Python, assignment does NOT copy, it only creates a reference.
a = [10, 20, [30, 40]]
b = a
b[0] = 42
print(a)
print(b)
```

```
[42, 20, [30, 40]]
[42, 20, [30, 40]]
```

[13]:
```
# Making a shallow copy.
a = [10, 20, [30, 40]]
b = copy.copy(a)
b[0] = 42
print(a)
print(b)
```

```
[10, 20, [30, 40]]
[42, 20, [30, 40]]
```

[14]:
```
# ...but:
a = [10, 20, [30, 40]]
b = copy.copy(a)
b[2][0] = 42
print(a)
print(b)
```

```
[10, 20, [42, 40]]
[10, 20, [42, 40]]
```

Shallow copy only copies at the highest level of the data structure!

[15]:
```
# Making a deep copy of a list of lists object.
a = [10, 20, [30, 40]]
b = copy.deepcopy(a)
b[2][0] = 42
print(a)
print(b)
```

```
[10, 20, [30, 40]]
[10, 20, [42, 40]]
```

gzip

- Provides tools for reading and writing GZIP archives.
- Remark: The standard library supports other formats too (e.g. BZ2, LZMA, ZIP, TAR).

[19]:
```
import gzip
```

[21]:
```
# Preparing a GZIP file.
text = 'Hello, Johnny! ' * 100
text
```

[21]:
```
'Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
```

```
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! '
```

[23]:
```
gzip.open('johnny.txt.gz', 'w').write(text.encode('utf-8'))
```

[23]:
```
1500
```

[25]:
```
# Reading a GZIP file.
gzip.open('johnny.txt.gz').read().decode('utf-8')
```

[25]:
```
'Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! '
```

[29]:
```
gzip.open('johnny.txt.gz', 'wt').write(text)
```

[29]:
```
1500
```

[30]:
```
gzip.open('johnny.txt.gz', 'rt').read()
```

[30]:
```
'Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
```

```
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny!
Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! Hello, Johnny! '
```

pickle

- Provides a solution for the serialization of Python data structures (transformation to bytes), and for deserialization.

[43]:
```python
import pickle
```

[46]:
```python
# Serializing a complex data structure to file.
data = [{'a': 1}, {'b': 2}]
pickle.dump(data, open('data.pkl', 'wb'))
```

[49]:
```python
# Deserialization.
data2 = pickle.load(open('data.pkl', 'rb'))
data2
```

[49]:
```
[{'a': 1}, {'b': 2}]
```

[51]:
```python
# Serialization to bytes.
b = pickle.dumps(data)
b
```

[51]:
```
b'\x80\x03]q\x00(}q\x01X\x01\x00\x00\x00aq\x02K\x01s}q\x03X\x01\x00\x00\x00bq\x0
4K\x02se.'
```

[52]:
```python
# Deserialization.
pickle.loads(b)
```

[52]:
```
[{'a': 1}, {'b': 2}]
```

[54]:
```python
# Two handy utility functions.

def to_pickle(obj, fname, protocol=4):
    '''Serialize object to file.'''
```

```
    pickle.dump(obj, open(fname, 'wb'), protocol)

def from_pickle(fname):
    '''Deserialize object from file.'''
    return pickle.load(open(fname, 'rb'))
```

[55]:
```
to_pickle(data, 'data.pkl')
```

[56]:
```
from_pickle('data.pkl')
```

[56]:
```
[{'a': 1}, {'b': 2}]
```

0.2 Exception handling

- Exception handling is a modern approach of error handling. It enables to handle the errors at the most appropriate location within the code.
- The earlier, error code based, method is less elegant. Assume that the error comes up deep in the function call stack. The error has to be handled at multiple locations (in the caller function, in the caller of the caller function etc.), which leads to code duplication or GOTO statements.
- Exceptions can be created with the raise statement, and they can be caught with the try statement.
- The hierarchy of the built-in exception types can be overviewed here.

[1]:
```
# Creating an exception.
for i in range(5):
    raise ValueError('foo')
    print('bar')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-b15b422be59f> in <module>
      1 # Creating an exception.
      2 for i in range(5):
----> 3     raise ValueError('foo')
      4     print('bar')

ValueError: foo
```

[2]:
```
1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-bc757c3fda29> in <module>
----> 1 1 / 0
```

```
ZeroDivisionError: division by zero
```

[3]:
```
# Catching an exception.
while True:
    try:
        x = float(input('x: '))
        y = float(input('y: '))
        z = x / y
        print(z)
        break
    except ZeroDivisionError:
        print('y should not be zero!')
    except ValueError:
        print('x and y should be float!')
```

```
x: apple
x and y should be float!
x: 3
y: 0
y should not be zero!
x: 3
y: 2
1.5
```

0.3 Debugging

[4]:
```
# First step: ALWAYS read the error message! :-)
1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-4-90b8ef851ca4> in <module>
      1 # First step: ALWAYS read the error message! :-)
----> 2 1 / 0

ZeroDivisionError: division by zero
```

[5]:
```
# Example for an erroneous function.
def calc_avg(list_of_lists):
    merged = []
    for lst in list_of_lists:
        merged.append(lst)
    return sum(merged) / len(merged)
```

```
data = [[3, 10, 5], [4, 8], [1, 8, 5]]
calc_avg(data)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-9a51e7c2dfa3> in <module>
      7
      8 data = [[3, 10, 5], [4, 8], [1, 8, 5]]
----> 9 calc_avg(data)

<ipython-input-5-9a51e7c2dfa3> in calc_avg(list_of_lists)
      4     for lst in list_of_lists:
      5         merged.append(lst)
----> 6     return sum(merged) / len(merged)
      7
      8 data = [[3, 10, 5], [4, 8], [1, 8, 5]]

TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

[6]:
```
# Find the error using the %debug command!
%debug
```

```
> <ipython-input-5-9a51e7c2dfa3>(6)calc_avg()
      4     for lst in
list_of_lists:
      5         merged.append(lst)
----> 6     return
sum(merged) /
len(merged)
      7
      8 data =
[[3, 10,
5], [4,
8], [1,
8,
5]]

ipdb> print(merged)
[[3, 10, 5], [4, 8], [1, 8, 5]]
ipdb> print(lst)
[1, 8, 5]
ipdb> q
```

[7]:
```
# The corrected version of the function.
def calc_avg(list_of_lists):
    merged = []
```

```
    for lst in list_of_lists:
        merged.extend(lst)
    return sum(merged) / len(merged)

data = [[3, 10, 5], [4, 8], [1, 8, 5]]
calc_avg(data)
```

[7]: 5.5

09_en

May 20, 2022

0.1 NumPy

NumPy is a low level, mathematical package for numerical computations.

- Its fundamental data structure is the n-dimensional array.
- It was written in C. The usual array operations are implemented efficiently.
- Among others, it contains submodules for linear algebra and random number generation.
- Several higher level packages (e.g. scipy, matplotlib, pandas, scikit-learn) are based on it.

NumPy is an external package, there are multiple methods to install it, for example: - `pip install numpy --user` - `sudo apt-get install python3-numpy` - `conda install numpy`

[14]:
```
# Importing NumPy under the name np.
import numpy as np
```

[15]:
```
# Querying the version number.
np.__version__
```

[15]:
```
'1.18.1'
```

Creating arrays

[16]:
```
# Create a 1-dimensional array of integers.
a = np.array([2, 8, 4, 3])
```

[17]:
```
a
```

[17]:
```
array([2, 8, 4, 3])
```

[18]:
```
# Type of the array object.
type(a)
```

[18]:
```
numpy.ndarray
```

[19]:
```
# Number of dimensions.
a.ndim
```

[19]:
```
1
```

[20]:
```
# Size of dimensions.
a.shape
```

[20]:
```
(4,)
```

[21]:
```
# The data type of the array elements.
# Arrays in are homogenous in NumPy (except the object array).
a.dtype
```

[21]:
```
dtype('int64')
```

[23]:
```
# Create a 2-dimensional array of floats.
b = np.array([[1.5, 2, 3], [5.5, 10, 2]])
```

[24]:
```
b
```

[24]:
```
array([[ 1.5,  2. ,  3. ],
       [ 5.5, 10. ,  2. ]])
```

[25]:
```
# Number and size of dimensions, data type.
b.ndim
```

[25]:
```
2
```

[26]:
```
b.shape
```

[26]:
```
(2, 3)
```

[27]:
```
b.dtype
```

[27]:
```
dtype('float64')
```

[29]:
```
# Specifying the data type of elements, example 1.
np.array([4, 5, 6, 7], dtype='uint8')
```

[29]:
```
array([4, 5, 6, 7], dtype=uint8)
```

[30]:
```
# Specifying the data type of elements, example 2.
np.array([[1, 0], [0, 2]], dtype='float32')
```

[30]:
```
array([[1., 0.],
       [0., 2.]], dtype=float32)
```

[31]:
```
# Loading an array from text file.
np.genfromtxt('matrix.txt')
```

[31]:
```
array([[0., 1., 1., 0., 1., 0., 1., 1., 0., 1.],
       [0., 0., 1., 0., 1., 1., 0., 1., 0., 1.],
       [0., 0., 1., 0., 0., 0., 1., 1., 0., 0.],
       [0., 1., 0., 0., 1., 0., 1., 1., 0., 0.],
       [1., 0., 1., 1., 0., 0., 1., 0., 1., 1.],
       [1., 0., 1., 0., 0., 1., 1., 0., 1., 0.],
       [1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
       [0., 0., 0., 0., 0., 1., 0., 1., 0., 1.],
       [1., 1., 0., 1., 0., 1., 1., 1., 0., 0.],
       [1., 0., 1., 0., 1., 0., 0., 1., 0., 1.]])
```

[32]:
```
# Create an array of zeros, example 1.
np.zeros(10)
```

[32]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[33]:
```
# Create an array of zeros, example 2.
np.zeros((2, 3), dtype='int32')
```

[33]:
```
array([[0, 0, 0],
       [0, 0, 0]], dtype=int32)
```

[34]:
```
# Create an array of ones, example 1.
np.ones((4, 7))
```

[34]:
```
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

[36]:
```
# Create an array of ones, example 2.
np.ones((2, 2, 2), dtype='float32')
```

[36]:
```
array([[[1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.]]], dtype=float32)
```

[37]:
```
np.ones((2, 2, 2, 2), dtype='float32')
```

[37]:
```
array([[[[1., 1.],
         [1., 1.]],

        [[1., 1.],
         [1., 1.]]],
```

```
[[[1., 1.],
  [1., 1.]],

 [[1., 1.],
  [1., 1.]]], dtype=float32)
```

[38]:
```
# Create an identity matrix.
np.eye(5)
```

[38]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

[40]:
```
# Create a range by specifying the step size.
np.arange(-2, 2, 0.1)
```

[40]:
```
array([-2.00000000e+00, -1.90000000e+00, -1.80000000e+00, -1.70000000e+00,
       -1.60000000e+00, -1.50000000e+00, -1.40000000e+00, -1.30000000e+00,
       -1.20000000e+00, -1.10000000e+00, -1.00000000e+00, -9.00000000e-01,
       -8.00000000e-01, -7.00000000e-01, -6.00000000e-01, -5.00000000e-01,
       -4.00000000e-01, -3.00000000e-01, -2.00000000e-01, -1.00000000e-01,
        1.77635684e-15,  1.00000000e-01,  2.00000000e-01,  3.00000000e-01,
        4.00000000e-01,  5.00000000e-01,  6.00000000e-01,  7.00000000e-01,
        8.00000000e-01,  9.00000000e-01,  1.00000000e+00,  1.10000000e+00,
        1.20000000e+00,  1.30000000e+00,  1.40000000e+00,  1.50000000e+00,
        1.60000000e+00,  1.70000000e+00,  1.80000000e+00,  1.90000000e+00])
```

[41]:
```
# Create a range by specifying the number of elements.
np.linspace(-2, 2, 7)
```

[41]:
```
array([-2.        , -1.33333333, -0.66666667,  0.        ,  0.66666667,
        1.33333333,  2.        ])
```

[42]:
```
# Concatenating vectors.
a = np.array([2, 3, 4])
b = np.array([10, 20])
np.concatenate([a, b])
```

[42]:
```
array([ 2,  3,  4, 10, 20])
```

[44]:
```
# Stacking matrices horizontally.
a = np.array([
    [2, 3, 4],
    [5, 6, 7]
```

```
])
np.hstack([a, a])
```

[44]:
```
array([[2, 3, 4, 2, 3, 4],
       [5, 6, 7, 5, 6, 7]])
```

[46]:
```
# Stacking matrices vertically.
np.vstack([a, a, a])
```

[46]:
```
array([[2, 3, 4],
       [5, 6, 7],
       [2, 3, 4],
       [5, 6, 7],
       [2, 3, 4],
       [5, 6, 7]])
```

Elements ans subarrays

[47]:
```
# Let's create an example matrix!
a = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
```

[48]:
```
# Select an element (indexing starts from 0)
a[0, 1] # 0 is the row index, 1 is the column index
```

[48]:
```
2
```

[51]:
```
# ...the following is done under the hood:
np.ndarray.__getitem__(a, (0, 1))
```

[51]:
```
2
```

[52]:
```
# Selecting a full row.
a[1, :]
```

[52]:
```
array([4, 5, 6])
```

[53]:
```
# We could also do it this way.
a[1]
```

[53]:
```
array([4, 5, 6])
```

[54]:
```
# The selected row is a 1-dimensional array.
a[1].ndim
```

[54]:
```
1
```

[55]:
```
# Selecting a column.
a[:, 2]
```

[55]:
```
array([3, 6])
```

[56]:
```
# Selecting a subarray.
a[:, :2]
```

[56]:
```
array([[1, 2],
       [4, 5]])
```

[58]:
```
# Selecting columns with the given indices.
a[:, [1, 0, 1, 0]]
```

[58]:
```
array([[2, 1, 2, 1],
       [5, 4, 5, 4]])
```

[60]:
```
# Selecting elements based on a logical condition.
a[a > 2]
```

[60]:
```
array([3, 4, 5, 6])
```

[61]:
```
# The elements of the array can be modified.
a[0, 0] = 100
a
```

[61]:
```
array([[100,   2,   3],
       [  4,   5,   6]])
```

[63]:
```
# Modifying a column.
a[:, 1] = [30, 40]
a
```

[63]:
```
array([[100,  30,   3],
       [  4,  40,   6]])
```

Array operations

[64]:
```
# Let's create 2 example arrays!
a = np.array([2, 3, 4])
b = np.array([1, 2, 3])
```

[65]:
```
# Elementwise addition.
a + b
```

[65]:
```
array([3, 5, 7])
```

[66]:
```
# Elementwise subtraction.
a - b
```

[66]:
```
array([1, 1, 1])
```

[67]:
```
# Elementwise multiplication.
a * b
```

[67]:
```
array([ 2,  6, 12])
```

[68]:
```
# Elementwise division.
a / b
```

[68]:
```
array([2.        , 1.5       , 1.33333333])
```

[69]:
```
# Elementwise integer division.
a // b
```

[69]:
```
array([2, 1, 1])
```

[70]:
```
# Elementwise exponentiation.
a**b
```

[70]:
```
array([ 2,  9, 64])
```

[71]:
```
# The operation is not necessarily doable.
np.array([1, 2]) + np.array([1, 2, 3])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-71-9725c2359035> in <module>
      1 # The operation is not necessarily doable.
----> 2 np.array([1, 2]) + np.array([1, 2, 3])

ValueError: operands could not be broadcast together with shapes (2,) (3,)
```

[72]:
```
# Display again the array "a"!
a
```

[72]:
```
array([2, 3, 4])
```

[73]:
```
# Elementwise functions (exp, log, sin, cos, ...).
np.cos(a)
```

[73]:
```
array([-0.41614684, -0.9899925 , -0.65364362])
```

[74]:
```
np.log(a)
```

[74]:
```
array([0.69314718, 1.09861229, 1.38629436])
```

[75]:
```
# Statistical operations (min, max, sum, mean, std).
c = np.array([
    [2, 3, 4],
    [5, 6, 10]
])
```

[76]:
```
c.min()
```

[76]:
```
2
```

[77]:
```
c.sum()
```

[77]:
```
30
```

[78]:
```
c.mean()
```

[78]:
```
5.0
```

[79]:
```
c.std() # standard deviation
```

[79]:
```
2.581988897471611
```

[81]:
```
# Columnwise statistics.
# We aggregate along the 0-th dimension i.e. rows, therefore this dimension␣
 ↪will disappear.
c.mean(axis=0)
```

[81]:
```
array([3.5, 4.5, 7. ])
```

[82]:
```
# Rowwise statistics.
# We aggregate along the 1-st dimension i.e. columns, therefore this dimension␣
 ↪will disappear.
c.sum(axis=1)
```

[82]:
```
array([ 9, 21])
```

[85]:
```
# Exercise: Create a 3×3 NumPy array of logical True values.

# solution 1:
np.array([[True] * 3] * 3)
```

[85]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

[86]:
```
np.ones((3, 3), dtype='bool')
```

[86]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

[91]:
```
# Exercise: Print the odd values in the following 2-dimensional NumPy array!
a = np.array([
    [2, 4, 11],
    [5, 6, 7]
])
print(a[a % 2 == 1])
```

```
[11  5  7]
```

[95]:
```
# Exercise: Print the values greater than 3 in the following 2-dimensional␣
 ↪NumPy array!
a = np.array([
    [2, 4, 11],
    [5, 6, 7]
])
print(a[a > 3])
```

```
[ 4 11  5  6  7]
```

[100]:
```
# Exercise: Print the values greater than the average value in the following␣
 ↪NumPy array!
a = np.array([
    [2, 4, 11],
    [5, 6, 7]
])
print(a[a > a.mean()])
```

```
[11  6  7]
```

[102]:
```
# Type conversion.
a = np.array([2, 3, 4])
b = a.astype('float32')
b
```

[102]:
```
array([2., 3., 4.], dtype=float32)
```

[103]:
```
# Matrix transpose.
a = np.array([
    [2, 3, 4],
    [5, 6, 7]
])
a.T
```

[103]:
```
array([[2, 5],
       [3, 6],
       [4, 7]])
```

[104]:
```
# Transposition does not copy. It only creates a new view on the original data.
b = a.T
b[0, 0] = 100
a
```

[104]:
```
array([[100,   3,   4],
       [  5,   6,   7]])
```

[106]:
```
# ...if we want to create a new array:
a = np.array([
    [2, 3, 4],
    [5, 6, 7]
])
b = a.T.copy()
b[0, 0] = 100
print(a)
print(b)
```

```
[[2 3 4]
 [5 6 7]]
[[100   5]
 [  3   6]
 [  4   7]]
```

[107]:
```
# Create an example array of size 12!
a = np.arange(12)
a
```

[107]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

[108]:
```
# Reshaping to size 2×6.
a.reshape((2, 6))
```

[108]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

[109]:
```
# It is enough to specify only one dimension's size, we can write (-1) for the␣
↪other.
a.reshape((2, -1))
```

[109]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

[110]:
```
a.reshape((-1, 6))
```

[110]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

[111]:
```
# Reshaping to size 4×3.
a.reshape((4, -1))
```

[111]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

[112]:
```
# If the total element count cannot be 12, then we get an error.
a.reshape((-1, 5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-112-440c94d69f80> in <module>
      1 # If the total element count cannot be 12, then we get an error.
----> 2 a.reshape((-1, 5))

ValueError: cannot reshape array of size 12 into shape (5)
```

[113]:
```
# Reshaping to size 2×2×3.
a.reshape((2, 2, -1))
```

[113]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]]])
```

[4]:
```
# Assignment does not copy in NumPy.
a = np.array([2, 3, 4])
b = a
b[2] = 100
a
```

[4]:
```
array([  2,   3, 100])
```

[5]:
```
# We can copy using the copy method.
a = np.array([2, 3, 4])
b = a.copy()
b[0] = 200
a
```

[5]:
```
array([2, 3, 4])
```

[6]:
```
# Searching.
# Example: What are the indices of the elements less than 5?
a = np.array([10, 4, 2, 5, 6, 1])
# The index is used to get the element (instead of a tuple that has one element)
np.where(a < 5)[0]
```

[6]:
```
array([1, 2, 5])
```

[9]:
```
# Sorting in place.
a = np.array([10, 4, 2, 5, 6, 1])
a.sort()
a
```

[9]:
```
array([ 1,  2,  4,  5,  6, 10])
```

[10]:
```
# Sorting into a new array.
a = np.array([10, 4, 2, 5, 6, 1])
np.sort(a)
```

[10]:
```
array([ 1,  2,  4,  5,  6, 10])
```

[13]:
```
# Sorting into descending order.
a = np.array([10, 4, 2, 5, 6, 1])
np.sort(a)[::-1]
```

[13]:
```
array([10,  6,  5,  4,  2,  1])
```

[16]:
```
# The indices of the sorted elements in the original array.
a = np.array([10, 4, 2, 5, 6, 1])
idxs = np.argsort(a)
idxs
```

[16]:
```
array([5, 2, 1, 3, 4, 0])
```

[17]:
```
# Sorting the array using the index array.
a[idxs]
```

[17]:
```
array([ 1,  2,  4,  5,  6, 10])
```

[18]:
```
# NumPy provides a min, max, argmin and argmax function too.
a = np.array([10, 4, 2, 5, 6, 1])
a.min()
```

[18]:
```
1
```

[19]:
```
a.argmin()
```

[19]:
```
5
```

[22]:
```
# The sorting operations can be used rowwise and columnwise too.
b = np.array([
    [2, 6, 8],
    [3, 9, 7]
])

# Make a copy of the original matrix
c = b.copy()
print(c)

# Rowwise sorting (along axis 1)
c.sort(axis=1)
print(c)
```

```
[[2 6 8]
 [3 9 7]]
[[2 6 8]
 [3 7 9]]
```

[23]:
```
b.min(1) # rowwise minimum
```

[23]:
```
array([2, 3])
```

[24]:
```
b.max(0) # columnwise maximum
```

[24]:
```
array([3, 9, 8])
```

[25]:
```
# Scalar product of two vectors.
a = np.array([1, 2, 3])
b = np.array([2, 2, 2])
a @ b
```

[25]:
```
12
```

[26]:
```
# Matrix multiplication.
a = np.array([
    [2, 3, 4],
    [5, 6, 7]
])
a.T @ a
```

[26]:
```
array([[29, 36, 43],
       [36, 45, 54],
       [43, 54, 65]])
```

[27]:
```
a @ a.T
```

[27]:
```
array([[ 29,  56],
       [ 56, 110]])
```

Broadcasting

- Broadcasting is a mechanism to handle operands with different shape.
- Example:

```
A (4d array):      8 x 1 x 6 x 5
B (3d array):          7 x 1 x 5
Result (4d array): 8 x 7 x 6 x 5
```

[28]:
```
# Multiplying a vector by a scalar.
a = np.array([2, 3, 4])
b = 10
a * b
# a: 3
# b:  -
```

[28]:
```
array([20, 30, 40])
```

[29]:
```
# Example for non-broadcastable arrays.
a = np.array([2, 3, 4])
b = np.array([5, 6])
a + b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-36c10090aa64> in <module>
      2 a = np.array([2, 3, 4])
      3 b = np.array([5, 6])
----> 4 a + b
```

```
ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

[31]:
```
# Multiplying a matrix by a vector.
a = np.array([
    [2, 3, 4],
    [5, 6, 7]
])
b = np.array([1, 2, 3])
a * b

# a: 2 * 3
# b:     3
```

[31]:
```
array([[ 2,  6, 12],
       [ 5, 12, 21]])
```

[35]:
```
# Rowwise multiplication.
a = np.array([
    [2, 3, 4],
    [5, 6, 7]
])
b = np.array([1, 2])
(a.T * b).T
```

[35]:
```
array([[ 2,  3,  4],
       [10, 12, 14]])
```

0.2 Exercise: Univariate linear regression

The text file baseball.txt contains data about the height and weight of professional baseball players. Write a program that gives a linear model for predicting the weight from the height! Subtasks:

- Determine the model parameter that gives the lowest RMSE (root mean squared error)!
- Compute the model's RMSE and MAE (mean absolute error) on the training data set!

[40]:
```
data = np.genfromtxt('baseball.txt', delimiter=',')
data
```

[40]:
```
array([[188.,  82.],
       [188.,  98.],
       [183.,  95.],
       …,
       [190.,  93.],
       [190.,  86.],
```

```
       [185.,  88.]])
```

[43]:
```
# input vector
x = data[:, 0]

# target vector
y = data[:, 1]
```

[50]:
```
xm = x.mean()
ym = y.mean()
x -= xm
y -= ym
```

[51]:
```
x
```

[51]:
```
array([ 0.92836399,  0.92836399, -4.07163601, …,  2.92836399,
        2.92836399, -2.07163601])
```

[52]:
```
y
```

[52]:
```
array([-9.50338819,  6.49661181,  3.49661181, …,  1.49661181,
       -5.50338819, -3.50338819])
```

[56]:
```
# optimal model parameter
w = (x @ y) / (x @ x)
w
```

[56]:
```
0.8561919786085516
```

[63]:
```
# RMSE of the model on the data set
yhat = x * w
rmse = ((yhat - y)**2).mean()**0.5
rmse
```

[63]:
```
8.071205900903676
```

[69]:
```
# MAE of the model on the data set
mae = np.abs(yhat - y).mean()
mae
```

[69]:
```
6.398400208620017
```

[73]:
```
# let's predict the weight of player with height 180 cm
(180 - xm) * w + ym
```

[73]:
```
85.44871016095308
```

10_en

May 20, 2022

0.1 pandas

- pandas is a NumPy based data analysis tool. Many of its ideas were borrowed from the R language.
- pandas' fundamental data types are the DataFrame (table) and the Series (column).
- pandas can be viewed as an in-memory, column oriented database.

[1]:
```
# Importing pandas as pd.
import pandas as pd
```

[2]:
```
# The version number of pandas.
pd.__version__
```

[2]:
```
'0.25.3'
```

[12]:
```
# Creating a DataFrame from columns.
# The input is a dict, where the keys are the column names and the values are␣
 ↪the columns.
data = {
    'a': [10, 20, 30, 40],
    'b': ['xx', 'yy', 'zz', 'qq'],
    'c': [1.5, 2.6, 3.7, 4.8]
}
df1 = pd.DataFrame(data)
df1
```

[12]:

	a	b	c
0	10	xx	1.5
1	20	yy	2.6
2	30	zz	3.7
3	40	qq	4.8

[13]:
```
# The type of df1.
type(df1)
```

[13]:
```
pandas.core.frame.DataFrame
```

[14]:
```
# Column names.
df1.columns
```

[14]:
```
Index(['a', 'b', 'c'], dtype='object')
```

[15]:
```
# Iterating over column names.
for c in df1:
    print(c)
```

```
a
b
c
```

[16]:
```
# Number of rows.
len(df1)
```

[16]:
```
4
```

[17]:
```
# Shape of the DataFrame.
df1.shape
```

[17]:
```
(4, 3)
```

[18]:
```
# Summary information.
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
a    4 non-null int64
b    4 non-null object
c    4 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 224.0+ bytes
```

[20]:
```
# Basic statistics about the numerical columns.
df1.describe().T
```

[20]:

	count	mean	std	min	25%	50%	75%	max
a	4.0	25.00	12.909944	10.0	17.500	25.00	32.500	40.0
c	4.0	3.15	1.420094	1.5	2.325	3.15	3.975	4.8

[26]:
```
# Creating a DataFrame from rows.
# The input is a list of dicts, where each dict represents a row.
data = [
    {'a': 10, 'b': 'Joe', 'c': 1.5},
    {'a': 20, 'c': 1.5},
```

```
    {'a': 30, 'b': 'Tom', 'c': 2.5},
    {'a': 40, 'b': 'George', 'c': 5.5}
]
df2 = pd.DataFrame(data)
df2
```

[26]:

	a	b	c
0	10	Joe	1.5
1	20	NaN	1.5
2	30	Tom	2.5
3	40	George	5.5

[25]:
```
# !!!
import numpy as np
np.nan == np.nan
```

[25]:
```
False
```

[27]:
```
# Every DataFrame (and Series) contains an index.
# By default, the index starts from 0 and increases by 1.
df2.index
```

[27]:
```
RangeIndex(start=0, stop=4, step=1)
```

[30]:
```
# This can be overridden of course.
df3 = pd.DataFrame(data, index=['red', 'yellow', 'green', 'blue'])
df3
```

[30]:

	a	b	c
red	10	Joe	1.5
yellow	20	NaN	1.5
green	30	Tom	2.5
blue	40	George	5.5

[32]:
```
# Creating a Series, without specifying an index.
se1 = pd.Series([20, 30, 40])
se1
```

[32]:
```
0    20
1    30
2    40
dtype: int64
```

[33]:
```
# The type of the result.
type(se1)
```

[33]:
```
pandas.core.series.Series
```

[54]:
```
# Creating a Series with an index.
se2 = pd.Series([20, 30, 40], index=['a', 'b', 'c'])
se2
```

[54]:
```
a    20
b    30
c    40
dtype: int64
```

[36]:
```
# A column can be selected from a DataFrame using the [] operator.
df3['a']
```

[36]:
```
red       10
yellow    20
green     30
blue      40
Name: a, dtype: int64
```

[37]:
```
# ...or if the column name is a valid identifier, then the . operator can be␣
 ↪used too.
df3.a
```

[37]:
```
red       10
yellow    20
green     30
blue      40
Name: a, dtype: int64
```

[40]:
```
# Selecting multiple columns.
df3[['b', 'a', 'b', 'a']]
```

[40]:

	b	a	b	a
red	Joe	10	Joe	10
yellow	NaN	20	NaN	20
green	Tom	30	Tom	30
blue	George	40	George	40

[41]:
```
# If we index by a list of size 1, then the type of the result is DataFrame.
df3[['b']]
```

[41]:

	b
red	Joe
yellow	NaN
green	Tom
blue	George

[46]:
```
# Selecting rows from a DataFrame.
df3.loc['yellow']
```

[46]:
```
a     20
b    NaN
c    1.5
Name: yellow, dtype: object
```

[47]:
```
df3.loc[['yellow', 'green']]
```

[47]:
```
          a    b    c
yellow  20  NaN  1.5
green   30  Tom  2.5
```

[48]:
```
# Position based selection of rows.
df3.iloc[1]
```

[48]:
```
a     20
b    NaN
c    1.5
Name: yellow, dtype: object
```

[49]:
```
df3.iloc[[2, 3]]
```

[49]:
```
         a       b    c
green  30     Tom  2.5
blue   40  George  5.5
```

[55]:
```
# Selecting an item from a Series.
se2['b']
```

[55]:
```
30
```

[56]:
```
# Accessing the raw data.
se2.values
```

[56]:
```
array([20, 30, 40])
```

[57]:
```
df3.values
```

[57]:
```
array([[10, 'Joe', 1.5],
       [20, nan, 1.5],
       [30, 'Tom', 2.5],
       [40, 'George', 5.5]], dtype=object)
```

0.1.1 SELECT

[58]:
```
# Let's create an example DataFrame!
df = pd.DataFrame([
    {'student': 'John Doe', 'subject': 'Mathematics',  'grade': 5},
    {'student': 'John Doe', 'subject': 'History', 'grade': 2},
    {'student': 'Jane Smith', 'subject': 'Mathematics',  'grade': 3},
    {'student': 'Jane Smith', 'subject': 'Mathematics',  'grade': 5},
    {'student': 'Jane Smith', 'subject': 'History', 'grade': 4},
    {'student': 'Scunner Campbell', 'subject': 'Mathematics',  'grade': 1},
    {'student': 'Scunner Campbell', 'subject': 'Mathematics',  'grade': 2},
    {'student': 'Scunner Campbell', 'subject': 'History', 'grade': 5},
])
df
```

[58]:

	student	subject	grade
0	John Doe	Mathematics	5
1	John Doe	History	2
2	Jane Smith	Mathematics	3
3	Jane Smith	Mathematics	5
4	Jane Smith	History	4
5	Scunner Campbell	Mathematics	1
6	Scunner Campbell	Mathematics	2
7	Scunner Campbell	History	5

[70]:
```
# Logical condition column.
df['student'] == 'John Doe'
```

[70]:
```
0     True
1     True
2    False
3    False
4    False
5    False
6    False
7    False
Name: student, dtype: bool
```

[72]:
```
# All grades of John Doe.
df[df['student'] == 'John Doe']
```

[72]:

	student	subject	grade
0	John Doe	Mathematics	5
1	John Doe	History	2

[71]:
```
df[df['student'] == 'John Doe']['grade']
```

[71]:
```
0    5
1    2
Name: grade, dtype: int64
```

[69]:
```
# Grades better than 1 and worse than 5.
df[(df['grade'] > 1) & (df['grade'] < 5)]
```

[69]:

	student	subject	grade
1	John Doe	History	2
2	Jane Smith	Mathematics	3
4	Jane Smith	History	4
6	Scunner Campbell	Mathematics	2

0.1.2 GROUPBY

In pandas, the steps of grouping are as follows:

- **Splitting** the data into groups based on some criteria.
- **Applying** a function to each group independently.
- **Combining** the results into a data structure.

Out of these, the split step is the most straightforward. The splitting criterion is usually a column or a set of columns. The apply step is typically an aggregation (e.g. number of items in the group, number of unique values, sum, mean, minimum, maximum, first record, last record). If the apply step is an aggregation, then combining runs automatically, otherwise the programmer has to initiate it.

[73]:
```
# Let's use the previous DataFrame!
df
```

[73]:

	student	subject	grade
0	John Doe	Mathematics	5
1	John Doe	History	2
2	Jane Smith	Mathematics	3
3	Jane Smith	Mathematics	5
4	Jane Smith	History	4
5	Scunner Campbell	Mathematics	1
6	Scunner Campbell	Mathematics	2
7	Scunner Campbell	History	5

[75]:
```
# Group by subject.
gb = df.groupby('subject')
```

[76]:
```
# The type of the result.
type(gb)
```

[76]:
```
pandas.core.groupby.generic.DataFrameGroupBy
```

[77]:
```
# Number of records per subject.
gb.size()
```

[77]:
```
subject
History        3
Mathematics    5
dtype: int64
```

[78]:
```
# The same query, without using the gb variable.
df.groupby('subject').size()
```

[78]:
```
subject
History        3
Mathematics    5
dtype: int64
```

[80]:
```
# Average grade per subject.
df.groupby('subject').mean()
```

[80]:
```
               grade
subject
History     3.666667
Mathematics 3.200000
```

[82]:
```
# Extracting the grade column as a Series:
df.groupby('subject')['grade'].mean()
```

[82]:
```
subject
History        3.666667
Mathematics    3.200000
Name: grade, dtype: float64
```

[85]:
```
# Subject averages for every student.
df.groupby(['student', 'subject'])['grade'].mean()
```

[85]:
```
student           subject
Jane Smith        History        4.0
                  Mathematics    4.0
John Doe          History        2.0
                  Mathematics    5.0
Scunner Campbell  History        5.0
                  Mathematics    1.5
Name: grade, dtype: float64
```

[86]:
```
# Changing the index to 2 ordinary columns.
df.groupby(['student', 'subject'])['grade'].mean().reset_index()
```

[86]:

	student	subject	grade
0	Jane Smith	History	4.0
1	Jane Smith	Mathematics	4.0
2	John Doe	History	2.0
3	John Doe	Mathematics	5.0
4	Scunner Campbell	History	5.0
5	Scunner Campbell	Mathematics	1.5

0.2 Example queries on the pl.txt data set

[1]:
```
import pandas as pd
```

[4]:
```
pd.__version__
```

[4]:
```
'1.1.4'
```

[2]:
```
# Loading pl.txt into a DataFrame.
names = ['round', 'hteam', 'ateam', 'hgoals', 'agoals']
df = pd.read_csv('pl.txt', sep='\t', skiprows=6, names=names)
df
```

[2]:

	round	hteam	ateam	hgoals	agoals
0	1	Blackburn Rovers	Wolverhampton Wanderers	1	2
1	1	Fulham FC	Aston Villa	0	0
2	1	Liverpool FC	Sunderland AFC	1	1
3	1	Queens Park Rangers	Bolton Wanderers	0	4
4	1	Wigan Athletic	Norwich City	1	1
..	…	…	…	…	…
375	38	Sunderland AFC	Manchester United	0	1
376	38	Swansea City	Liverpool FC	1	0
377	38	Tottenham Hotspur	Fulham FC	2	0
378	38	West Bromwich Albion	Arsenal FC	2	3
379	38	Wigan Athletic	Wolverhampton Wanderers	3	2

[380 rows x 5 columns]

[3]:
```
# Add "number of goals" column to the DataFrame.
df['goals'] = df['hgoals'] + df['agoals']
df
```

[3]:

	round	hteam	ateam	hgoals	agoals	\
0	1	Blackburn Rovers	Wolverhampton Wanderers	1	2	
1	1	Fulham FC	Aston Villa	0	0	
2	1	Liverpool FC	Sunderland AFC	1	1	
3	1	Queens Park Rangers	Bolton Wanderers	0	4	
4	1	Wigan Athletic	Norwich City	1	1	

```
..     …                    …                        …     …     …
375    38        Sunderland AFC        Manchester United     0     1
376    38          Swansea City             Liverpool FC     1     0
377    38    Tottenham Hotspur                Fulham FC     2     0
378    38  West Bromwich Albion              Arsenal FC     2     3
379    38        Wigan Athletic  Wolverhampton Wanderers     3     2

     goals
0        3
1        0
2        2
3        4
4        2
..     …
375      1
376      1
377      2
378      5
379      5

[380 rows x 6 columns]
```

[10]:
```
# Number of goals in round 13.
df[df['round'] == 13]['goals'].sum()
```

[10]:
```
25
```

[17]:
```
# Which round had the highest number of goals?
se = df.groupby('round')['goals'].sum()
print(se.idxmax(), se.max())
```

```
10 39
```

[18]:
```
# Number of games per round.
df.groupby('round').size()
```

[18]:
```
round
1     10
2     10
3     10
4     10
5     10
6     10
7     10
8     10
9     10
10    10
```

```
11    10
12    10
13    10
14    10
15    10
16    10
17    10
18    10
19    10
20    10
21    10
22    10
23    10
24    10
25    10
26    10
27    10
28    10
29    10
30    10
31    10
32    10
33    10
34    10
35    10
36    10
37    10
38    10
dtype: int64
```

[22]:
```
# Print the 10 rounds with the highest number of goals!
df.groupby('round')['goals'].sum().sort_values(ascending=False)[:10]
```

[22]:
```
round
10    39
5     38
25    36
36    35
22    35
31    35
7     34
38    32
8     32
24    31
Name: goals, dtype: int64
```

[29]:
```
# What percentage of the games had at least 1 goal?
(df['goals'] > 0).sum() / len(df) * 100
```

[29]:
```
92.89473684210526
```

[32]:
```
(df['goals'] > 0).mean() * 100
```

[32]:
```
92.89473684210526
```

[37]:
```
# What was the game with the highest number of goals?
df.loc[df['goals'].idxmax()]
```

[37]:
```
round                     3
hteam     Manchester United
ateam             Arsenal FC
hgoals                    8
agoals                    2
goals                    10
Name: 29, dtype: object
```

[41]:
```
# Total number of goals scored by Manchester United?
team = 'Manchester United'
gh = df[df['hteam'] == team]['hgoals'].sum()
ga = df[df['ateam'] == team]['agoals'].sum()
gh + ga
```

[41]:
```
89
```

[50]:
```
# Top 5 teams, scoring the most goals.
se = df.groupby('hteam')['hgoals'].sum() + df.groupby('ateam')['agoals'].sum()
se.sort_values(ascending=False)[:5]
```

[50]:
```
hteam
Manchester City      93
Manchester United    89
Arsenal FC           74
Tottenham Hotspur    66
Chelsea FC           65
dtype: int64
```

0.3 Analyzing unicef.txt

unicef.txt contains nutrition data about under 5 population of the World. Every row corresponds to a survey, and the surveys are in chronological order per country. Write a program that reads data from unicef.txt, selects the most recent survey for each country, and displays the following statistics:

1. In what portion of the countries is the Underweight indicator higher than the Overweight indicator?
2. Which are the 5 countries with the highest Severe Wasting indicator?
3. Per United Nations Region, what is the ratio of countries with >20% Stunting?
4. Where is the value of Overweight the lowest among countries with Wasting and Stunting less that 10%?

[65]:
```
df0 = pd.read_csv('unicef.txt', sep='|', decimal=',')
df0
```

[65]:

	Country	United Nations Region	United Nations Sub-Region	World Bank Income Classification	Survey Year	Survey Sample (N)	Severe Wasting	Wasting	Stunting	Underweight	Overweight
0	AFGHANISTAN	Asia	Southern Asia	Low Income	1997	4846.0	NaN	18.2	53.2	44.9	6.5
1	AFGHANISTAN	Asia	Southern Asia	Low Income	2004	946.0	3.5	8.6	59.3	32.9	4.6
2	AFGHANISTAN	Asia	Southern Asia	Low Income	2013	21922.0	4.0	9.5	40.9	25.0	5.4
3	ALBANIA	Europe	Southern Europe	Upper Middle Income	1996-98	7642.0	NaN	8.1	20.4	7.1	9.5
4	ALBANIA	Europe	Southern Europe	Upper Middle Income	2000	1382.0	6.2	12.2	39.2	17.0	30.0
..	…	…	…	…	…	…	…	…	…	…	…
849	ZIMBABWE	Africa	Eastern Africa	Low Income	2005-06	5273.0	2.9	7.3	35.3	14.0	8.8
850	ZIMBABWE	Africa	Eastern Africa	Low Income	2009	6196.0	1.9	3.8	35.1	12.6	3.5
851	ZIMBABWE	Africa	Eastern Africa	Low Income	2010-11	5414.0	0.8	3.2	32.2	10.2	5.8
852	ZIMBABWE	Africa	Eastern Africa	Low Income	2014	9591.0	0.9	3.4	27.6	11.3	3.6
853	ZIMBABWE	Africa	Eastern Africa	Low Income	2015	6380.0	1.1	3.3	27.1	8.5	5.6

```
                                                  Source                 Notes  \
0     Afghanistan 1997 multiple indicator baseline …   Converted estimates
1    Summary report of the national nutrition surve…                   NaN
2          Afghanistan National Nutrition Survey 2013.  (pending reanalysis)
3    National study on nutrition in Albania. Institt…   Converted estimates
4    Multiple indicator cluster survey report Alban…                   NaN
..                                                 …                   …
849  Zimbabwe demographic and health survey 2005-06…                   NaN
850  Zimbabwe multiple indicator monitoring survey …                   NaN
851  Zimbabwe demographic and health survey 2010-11…                   NaN
852  Zimbabwe Multiple Indicator Cluster Survey 201…                   NaN
853  Zimbabwe Demographic and Health Survey 2015, N…                   NaN

     U5 Population ('000s)
0                 3637.632
1                 4667.487
2                 5235.867
3                  307.887
4                  278.753
..                     …
849               1950.476
850               2160.621
851               2225.702
852               2466.295
853               2505.484

[854 rows x 14 columns]
```

[66]:
```
df0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 854 entries, 0 to 853
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Country                          854 non-null    object
 1   United Nations Region            854 non-null    object
 2   United Nations Sub-Region        854 non-null    object
 3   World Bank Income Classification 854 non-null    object
 4   Survey Year                      854 non-null    object
 5   Survey Sample (N)                826 non-null    float64
 6   Severe Wasting                   636 non-null    float64
 7   Wasting                          810 non-null    float64
 8   Stunting                         820 non-null    float64
 9   Underweight                      836 non-null    float64
 10  Overweight                       721 non-null    float64
```

```
 11  Source                          854 non-null    object
 12  Notes                           303 non-null    object
 13  U5 Population ('000s)           854 non-null    float64
dtypes: float64(7), object(7)
memory usage: 93.5+ KB
```

[67]:
```
df0.describe().T
```

[67]:

	count	mean	std	min	25%	\
Survey Sample (N)	826.0	286560.445521	2.143035e+06	267.0	2713.500	
Severe Wasting	636.0	2.317925	2.059254e+00	0.0	0.800	
Wasting	810.0	7.105309	5.181751e+00	0.0	2.925	
Stunting	820.0	29.189634	1.597603e+01	0.0	16.875	
Underweight	836.0	15.956579	1.274767e+01	0.0	4.900	
Overweight	721.0	6.530791	4.583190e+00	0.0	3.100	
U5 Population ('000s)	854.0	6226.739792	1.687893e+04	1.0	523.585	

	50%	75%	max
Survey Sample (N)	5032.000	11740.50000	4.983350e+07
Severe Wasting	1.700	3.00000	1.290000e+01
Wasting	5.950	9.97500	2.530000e+01
Stunting	29.200	40.50000	7.360000e+01
Underweight	13.150	23.40000	6.680000e+01
Overweight	5.500	8.80000	3.000000e+01
U5 Population ('000s)	1770.243	4181.01375	1.332297e+05

[68]:
```
# Select the most recent survey per country.
df = df0.groupby('Country').last()
df
```

[68]:

Country	United Nations Region	\
AFGHANISTAN	Asia	
ALBANIA	Europe	
ALGERIA	Africa	
ANGOLA	Africa	
ARGENTINA	Latin America and the Caribbean	
…	…	
VENEZUELA (BOLIVARIAN REPUBLIC OF)	Latin America and the Caribbean	
VIET NAM	Asia	
YEMEN	Asia	
ZAMBIA	Africa	
ZIMBABWE	Africa	

Country	United Nations Sub-Region	\
AFGHANISTAN	Southern Asia	

ALBANIA	Southern Europe	
ALGERIA	Northern Africa	
ANGOLA	Middle Africa	
ARGENTINA	South America	
…	…	
VENEZUELA (BOLIVARIAN REPUBLIC OF)	South America	
VIET NAM	South-Eastern Asia	
YEMEN	Western Asia	
ZAMBIA	Eastern Africa	
ZIMBABWE	Eastern Africa	

Country	World Bank Income Classification	\
AFGHANISTAN	Low Income	
ALBANIA	Upper Middle Income	
ALGERIA	Upper Middle Income	
ANGOLA	Lower Middle Income	
ARGENTINA	High Income	
…	…	
VENEZUELA (BOLIVARIAN REPUBLIC OF)	Upper Middle Income	
VIET NAM	Lower Middle Income	
YEMEN	Low Income	
ZAMBIA	Lower Middle Income	
ZIMBABWE	Low Income	

Country	Survey Year	Survey Sample (N)	\
AFGHANISTAN	2013	21922.0	
ALBANIA	2017-18	2367.0	
ALGERIA	2012-13	13860.0	
ANGOLA	2015-16	7468.0	
ARGENTINA	2004-05	999999.0	
…	…	…	
VENEZUELA (BOLIVARIAN REPUBLIC OF)	2009	242775.0	
VIET NAM	2015	98447.0	
YEMEN	2013	14624.0	
ZAMBIA	2013-14	12877.0	
ZIMBABWE	2015	6380.0	

Country	Severe Wasting	Wasting	Stunting	\
AFGHANISTAN	4.0	9.5	40.9	
ALBANIA	0.5	1.6	11.3	
ALGERIA	1.4	4.1	11.7	
ANGOLA	1.1	4.9	37.6	
ARGENTINA	0.2	1.2	8.2	
…	…	…	…	

```
VENEZUELA (BOLIVARIAN REPUBLIC OF)             NaN      4.1      13.4
VIET NAM                                       1.4      6.4      24.6
YEMEN                                          5.4     16.4      46.4
ZAMBIA                                         2.5      6.2      40.0
ZIMBABWE                                       1.1      3.3      27.1

                                  Underweight  Overweight  \
Country
AFGHANISTAN                              25.0         5.4
ALBANIA                                   1.5        16.4
ALGERIA                                   3.0        12.4
ANGOLA                                   19.0         3.4
ARGENTINA                                 2.3         9.9
…                                         …           …
VENEZUELA (BOLIVARIAN REPUBLIC OF)        2.9         6.4
VIET NAM                                 14.1         5.3
YEMEN                                    39.9         2.5
ZAMBIA                                   14.9         6.2
ZIMBABWE                                  8.5         5.6

Source  \
Country
AFGHANISTAN                              Afghanistan National Nutrition Survey
2013.
ALBANIA                           Albania Demographic and Health Survey
2017-18…
ALGERIA                           République Algérienne Démocratiqe et
Populaire…
ANGOLA                            Inquérito de Indicadores Múltiplos e de
Saúde …
ARGENTINA                         Nutrition status in Argentinean children 6
to …
…
…
VENEZUELA (BOLIVARIAN REPUBLIC OF)  Ficha técnica: Evaluación antropométrica
nutri…
VIET NAM                          Nutrition surveillance profiles 2015. Hanoi,
V…
YEMEN                             Yemen National Health and Demographic Survey
2…
ZAMBIA                            Zambia demographic and health Survey
2013-14. …
ZIMBABWE                          Zimbabwe Demographic and Health Survey 2015,
N…

                                                                        Notes
\
```

```
Country
AFGHANISTAN                                          (pending reanalysis)
ALBANIA                                               Converted estimates
ALGERIA                                               Converted estimates
ANGOLA                                                                NaN
ARGENTINA                                             Converted estimates
…                                                                       …
VENEZUELA (BOLIVARIAN REPUBLIC OF)                    Converted estimates
VIET NAM                          Surveillance data (updated report 11/1/2014)
YEMEN                                                 Converted estimates
ZAMBIA                                                 Converted estimate
ZIMBABWE                                                    Age-adjusted;

                                  U5 Population ('000s)
Country
AFGHANISTAN                                    5235.867
ALBANIA                                         177.175
ALGERIA                                        4340.456
ANGOLA                                         5277.122
ARGENTINA                                      3594.876
…                                                 …
VENEZUELA (BOLIVARIAN REPUBLIC OF)             2941.185
VIET NAM                                       7752.861
YEMEN                                          3895.949
ZAMBIA                                         2626.277
ZIMBABWE                                       2505.484

[152 rows x 13 columns]
```

[72]:
```
# 1. In what portion of the countries is the Underweight indicator higher than␣
 ↪the Overweight indicator?
(df['Underweight'] > df['Overweight']).mean()
```

[72]:
```
0.506578947368421
```

[77]:
```
# 2. Which are the 5 countries with the highest Severe Wasting indicator?
df['Severe Wasting'].sort_values(ascending=False)[:5]
```

[77]:
```
Country
SOUTH SUDAN         11.9
DJIBOUTI             9.2
INDIA                7.7
INDONESIA            6.7
PAPUA NEW GUINEA     6.4
Name: Severe Wasting, dtype: float64
```

[83]:
```
# ...alternatively:
df.sort_values('Severe Wasting', ascending=False)[:5]
```

[83]:

Country	United Nations Region	United Nations Sub-Region	World Bank Income Classification	Survey Year	Survey Sample (N)	Severe Wasting	Wasting	Stunting	Underweight	Overweight	Source	Notes
SOUTH SUDAN	Africa	Eastern Africa	Low Income	2010	6390.0	11.9	24.3	31.3	29.1	5.8	South Sudan Household Survey 2010. Final Repor…	NaN
DJIBOUTI	Africa	Eastern Africa	Lower Middle Income	2012	3153.0	9.2	21.6	33.5	29.9	8.1	Enquête djiboutienne à indicateurs multiples (…	Converted estimates
INDIA	Asia	Southern Asia	Lower Middle Income	2015-16	230448.0	7.7	20.8	37.9	36.3	2.4	National Family Health Survey (NFHS-4), 2015-1…	
INDONESIA	Asia	South-Eastern Asia	Lower Middle Income	2013	75232.0	6.7	13.5	36.4	19.9	11.5	National report on basic health research, RISK…	
PAPUA NEW GUINEA	Oceania	Melanesia	Lower Middle Income	2009-11	945911.0	6.4	14.1	49.5	27.8	13.7	2009-2010 Papua New Guinea household income an…	

```
INDIA                                      (pending reanalysis)
INDONESIA                                   Converted estimates
PAPUA NEW GUINEA    Adjusted NR to NA; 85% of total population

                  U5 Population ('000s)
Country
SOUTH SUDAN                    1657.580
DJIBOUTI                         99.067
INDIA                        121415.293
INDONESIA                     24249.408
PAPUA NEW GUINEA                983.166
```

[89]:
```
# 3. Per United Nations Region, what is the ratio of countries with >20%␣
↪Stunting?
# df.groupby('United Nations Region')
(df['Stunting'] > 20).groupby(df['United Nations Region']).mean()
```

[89]:
```
United Nations Region
Africa                             0.833333
Asia                               0.377778
Europe                             0.076923
Latin America and the Caribbean    0.148148
Northern America                   0.000000
Oceania                            0.545455
Name: Stunting, dtype: float64
```

[99]:
```
# 4. Where is the value of Overweight the lowest among countries with Wasting␣
↪and Stunting less that 10%?
df2 = df[(df['Wasting'] < 10) & (df['Stunting'] < 10)]
df2['Overweight'].idxmin()
```

[99]:
```
'JAPAN'
```

0.4 FIFA'19 Players

fifa19.csv.gz contains data about players in FIFA'19.

[103]:
```
# Load the data into a DataFrame!
df = pd.read_csv('fifa19.csv.gz')
del df['Unnamed: 0'] # remove the column 'Unnamed: 0'
df
```

[103]:
```
            ID                 Name  Age  \
0       158023             L. Messi   31
1        20801   Cristiano Ronaldo   33
2       190871           Neymar Jr   26
```

```
3      193080              De Gea   27
4      192985        K. De Bruyne   27
…       …                   …  …
18202  238813        J. Lundstram   19
18203  243165  N. Christoffersson   19
18204  241638           B. Worman   16
18205  246268      D. Walker-Rice   17
18206  246269           G. Nugent   16

                                                  Photo Nationality  \
0      https://cdn.sofifa.org/players/4/19/158023.png   Argentina
1       https://cdn.sofifa.org/players/4/19/20801.png    Portugal
2      https://cdn.sofifa.org/players/4/19/190871.png      Brazil
3      https://cdn.sofifa.org/players/4/19/193080.png       Spain
4      https://cdn.sofifa.org/players/4/19/192985.png     Belgium
…                                                  …           …
18202  https://cdn.sofifa.org/players/4/19/238813.png     England
18203  https://cdn.sofifa.org/players/4/19/243165.png      Sweden
18204  https://cdn.sofifa.org/players/4/19/241638.png     England
18205  https://cdn.sofifa.org/players/4/19/246268.png     England
18206  https://cdn.sofifa.org/players/4/19/246269.png     England

                                     Flag  Overall  Potential  \
0      https://cdn.sofifa.org/flags/52.png       94         94
1      https://cdn.sofifa.org/flags/38.png       94         94
2      https://cdn.sofifa.org/flags/54.png       92         93
3      https://cdn.sofifa.org/flags/45.png       91         93
4       https://cdn.sofifa.org/flags/7.png       91         92
…                                       …      …         …
18202  https://cdn.sofifa.org/flags/14.png       47         65
18203  https://cdn.sofifa.org/flags/46.png       47         63
18204  https://cdn.sofifa.org/flags/14.png       47         67
18205  https://cdn.sofifa.org/flags/14.png       47         66
18206  https://cdn.sofifa.org/flags/14.png       46         66

                      Club                                        Club Logo  \
0              FC Barcelona    https://cdn.sofifa.org/teams/2/light/241.png
1                  Juventus     https://cdn.sofifa.org/teams/2/light/45.png
2      Paris Saint-Germain      https://cdn.sofifa.org/teams/2/light/73.png
3        Manchester United      https://cdn.sofifa.org/teams/2/light/11.png
4          Manchester City      https://cdn.sofifa.org/teams/2/light/10.png
…                        …                                              …
18202      Crewe Alexandra    https://cdn.sofifa.org/teams/2/light/121.png
18203       Trelleborgs FF    https://cdn.sofifa.org/teams/2/light/703.png
18204     Cambridge United   https://cdn.sofifa.org/teams/2/light/1944.png
18205      Tranmere Rovers  https://cdn.sofifa.org/teams/2/light/15048.png
18206      Tranmere Rovers  https://cdn.sofifa.org/teams/2/light/15048.png
```

```
       … Composure Marking  StandingTackle SlidingTackle  GKDiving  \
0      …      96.0    33.0            28.0          26.0       6.0
1      …      95.0    28.0            31.0          23.0       7.0
2      …      94.0    27.0            24.0          33.0       9.0
3      …      68.0    15.0            21.0          13.0      90.0
4      …      88.0    68.0            58.0          51.0      15.0
…    …        …     …               …             …          …
18202  …      45.0    40.0            48.0          47.0      10.0
18203  …      42.0    22.0            15.0          19.0      10.0
18204  …      41.0    32.0            13.0          11.0       6.0
18205  …      46.0    20.0            25.0          27.0      14.0
18206  …      43.0    40.0            43.0          50.0      10.0

       GKHandling  GKKicking GKPositioning GKReflexes Release Clause
0            11.0       15.0          14.0        8.0         €226.5M
1            11.0       15.0          14.0       11.0         €127.1M
2             9.0       15.0          15.0       11.0         €228.1M
3            85.0       87.0          88.0       94.0         €138.6M
4            13.0        5.0          10.0       13.0         €196.4M
…            …          …             …          …            …
18202        13.0        7.0           8.0        9.0           €143K
18203         9.0        9.0           5.0       12.0           €113K
18204         5.0       10.0           6.0       13.0           €165K
18205         6.0       14.0           8.0        9.0           €143K
18206        15.0        9.0          12.0        9.0           €165K

[18207 rows x 88 columns]
```

[104]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18207 entries, 0 to 18206
Data columns (total 88 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        18207 non-null  int64
 1   Name                      18207 non-null  object
 2   Age                       18207 non-null  int64
 3   Photo                     18207 non-null  object
 4   Nationality               18207 non-null  object
 5   Flag                      18207 non-null  object
 6   Overall                   18207 non-null  int64
 7   Potential                 18207 non-null  int64
 8   Club                      17966 non-null  object
 9   Club Logo                 18207 non-null  object
 10  Value                     18207 non-null  object
```

```
11  Wage                      18207 non-null  object
12  Special                   18207 non-null  int64
13  Preferred Foot            18159 non-null  object
14  International Reputation  18159 non-null  float64
15  Weak Foot                 18159 non-null  float64
16  Skill Moves               18159 non-null  float64
17  Work Rate                 18159 non-null  object
18  Body Type                 18159 non-null  object
19  Real Face                 18159 non-null  object
20  Position                  18147 non-null  object
21  Jersey Number             18147 non-null  float64
22  Joined                    16654 non-null  object
23  Loaned From               1264 non-null   object
24  Contract Valid Until      17918 non-null  object
25  Height                    18159 non-null  object
26  Weight                    18159 non-null  object
27  LS                        16122 non-null  object
28  ST                        16122 non-null  object
29  RS                        16122 non-null  object
30  LW                        16122 non-null  object
31  LF                        16122 non-null  object
32  CF                        16122 non-null  object
33  RF                        16122 non-null  object
34  RW                        16122 non-null  object
35  LAM                       16122 non-null  object
36  CAM                       16122 non-null  object
37  RAM                       16122 non-null  object
38  LM                        16122 non-null  object
39  LCM                       16122 non-null  object
40  CM                        16122 non-null  object
41  RCM                       16122 non-null  object
42  RM                        16122 non-null  object
43  LWB                       16122 non-null  object
44  LDM                       16122 non-null  object
45  CDM                       16122 non-null  object
46  RDM                       16122 non-null  object
47  RWB                       16122 non-null  object
48  LB                        16122 non-null  object
49  LCB                       16122 non-null  object
50  CB                        16122 non-null  object
51  RCB                       16122 non-null  object
52  RB                        16122 non-null  object
53  Crossing                  18159 non-null  float64
54  Finishing                 18159 non-null  float64
55  HeadingAccuracy           18159 non-null  float64
56  ShortPassing              18159 non-null  float64
57  Volleys                   18159 non-null  float64
58  Dribbling                 18159 non-null  float64
```

```
 59  Curve                     18159 non-null  float64
 60  FKAccuracy                18159 non-null  float64
 61  LongPassing               18159 non-null  float64
 62  BallControl               18159 non-null  float64
 63  Acceleration              18159 non-null  float64
 64  SprintSpeed               18159 non-null  float64
 65  Agility                   18159 non-null  float64
 66  Reactions                 18159 non-null  float64
 67  Balance                   18159 non-null  float64
 68  ShotPower                 18159 non-null  float64
 69  Jumping                   18159 non-null  float64
 70  Stamina                   18159 non-null  float64
 71  Strength                  18159 non-null  float64
 72  LongShots                 18159 non-null  float64
 73  Aggression                18159 non-null  float64
 74  Interceptions             18159 non-null  float64
 75  Positioning               18159 non-null  float64
 76  Vision                    18159 non-null  float64
 77  Penalties                 18159 non-null  float64
 78  Composure                 18159 non-null  float64
 79  Marking                   18159 non-null  float64
 80  StandingTackle            18159 non-null  float64
 81  SlidingTackle             18159 non-null  float64
 82  GKDiving                  18159 non-null  float64
 83  GKHandling                18159 non-null  float64
 84  GKKicking                 18159 non-null  float64
 85  GKPositioning             18159 non-null  float64
 86  GKReflexes                18159 non-null  float64
 87  Release Clause            16643 non-null  object
dtypes: float64(38), int64(5), object(45)
memory usage: 12.2+ MB
```

[106]:
```
df.describe().T
```

[106]:

	count	mean	std	min	\
ID	18207.0	214298.338606	29965.244204	16.0	
Age	18207.0	25.122206	4.669943	16.0	
Overall	18207.0	66.238699	6.908930	46.0	
Potential	18207.0	71.307299	6.136496	48.0	
Special	18207.0	1597.809908	272.586016	731.0	
International Reputation	18159.0	1.113222	0.394031	1.0	
Weak Foot	18159.0	2.947299	0.660456	1.0	
Skill Moves	18159.0	2.361308	0.756164	1.0	
Jersey Number	18147.0	19.546096	15.947765	1.0	
Crossing	18159.0	49.734181	18.364524	5.0	
Finishing	18159.0	45.550911	19.525820	2.0	
HeadingAccuracy	18159.0	52.298144	17.379909	4.0	

ShortPassing	18159.0	58.686712	14.699495	7.0
Volleys	18159.0	42.909026	17.694408	4.0
Dribbling	18159.0	55.371001	18.910371	4.0
Curve	18159.0	47.170824	18.395264	6.0
FKAccuracy	18159.0	42.863153	17.478763	3.0
LongPassing	18159.0	52.711933	15.327870	9.0
BallControl	18159.0	58.369459	16.686595	5.0
Acceleration	18159.0	64.614076	14.927780	12.0
SprintSpeed	18159.0	64.726967	14.649953	12.0
Agility	18159.0	63.503607	14.766049	14.0
Reactions	18159.0	61.836610	9.010464	21.0
Balance	18159.0	63.966573	14.136166	16.0
ShotPower	18159.0	55.460047	17.237958	2.0
Jumping	18159.0	65.089432	11.820044	15.0
Stamina	18159.0	63.219946	15.894741	12.0
Strength	18159.0	65.311967	12.557000	17.0
LongShots	18159.0	47.109973	19.260524	3.0
Aggression	18159.0	55.868991	17.367967	11.0
Interceptions	18159.0	46.698276	20.696909	3.0
Positioning	18159.0	49.958478	19.529036	2.0
Vision	18159.0	53.400903	14.146881	10.0
Penalties	18159.0	48.548598	15.704053	5.0
Composure	18159.0	58.648274	11.436133	3.0
Marking	18159.0	47.281623	19.904397	3.0
StandingTackle	18159.0	47.697836	21.664004	2.0
SlidingTackle	18159.0	45.661435	21.289135	3.0
GKDiving	18159.0	16.616223	17.695349	1.0
GKHandling	18159.0	16.391596	16.906900	1.0
GKKicking	18159.0	16.232061	16.502864	1.0
GKPositioning	18159.0	16.388898	17.034669	1.0
GKReflexes	18159.0	16.710887	17.955119	1.0

	25%	50%	75%	max
ID	200315.5	221759.0	236529.5	246620.0
Age	21.0	25.0	28.0	45.0
Overall	62.0	66.0	71.0	94.0
Potential	67.0	71.0	75.0	95.0
Special	1457.0	1635.0	1787.0	2346.0
International Reputation	1.0	1.0	1.0	5.0
Weak Foot	3.0	3.0	3.0	5.0
Skill Moves	2.0	2.0	3.0	5.0
Jersey Number	8.0	17.0	26.0	99.0
Crossing	38.0	54.0	64.0	93.0
Finishing	30.0	49.0	62.0	95.0
HeadingAccuracy	44.0	56.0	64.0	94.0
ShortPassing	54.0	62.0	68.0	93.0
Volleys	30.0	44.0	57.0	90.0

Dribbling	49.0	61.0	68.0	97.0
Curve	34.0	48.0	62.0	94.0
FKAccuracy	31.0	41.0	57.0	94.0
LongPassing	43.0	56.0	64.0	93.0
BallControl	54.0	63.0	69.0	96.0
Acceleration	57.0	67.0	75.0	97.0
SprintSpeed	57.0	67.0	75.0	96.0
Agility	55.0	66.0	74.0	96.0
Reactions	56.0	62.0	68.0	96.0
Balance	56.0	66.0	74.0	96.0
ShotPower	45.0	59.0	68.0	95.0
Jumping	58.0	66.0	73.0	95.0
Stamina	56.0	66.0	74.0	96.0
Strength	58.0	67.0	74.0	97.0
LongShots	33.0	51.0	62.0	94.0
Aggression	44.0	59.0	69.0	95.0
Interceptions	26.0	52.0	64.0	92.0
Positioning	38.0	55.0	64.0	95.0
Vision	44.0	55.0	64.0	94.0
Penalties	39.0	49.0	60.0	92.0
Composure	51.0	60.0	67.0	96.0
Marking	30.0	53.0	64.0	94.0
StandingTackle	27.0	55.0	66.0	93.0
SlidingTackle	24.0	52.0	64.0	91.0
GKDiving	8.0	11.0	14.0	90.0
GKHandling	8.0	11.0	14.0	92.0
GKKicking	8.0	11.0	14.0	91.0
GKPositioning	8.0	11.0	14.0	90.0
GKReflexes	8.0	11.0	14.0	94.0

[110]:
```
# How many clubs are in the data set?
df['Club'].nunique()
```

[110]: 651

[116]:
```
# Which nationality is the most frequent?
df.groupby('Nationality').size().idxmax()
```

[116]: 'England'

[117]:
```
# What is the average age?
df['Age'].mean()
```

[117]: 25.122205745043114

[120]:
```
# Who is the oldest player?
df[df['Age'] == df['Age'].max()]
```

[120]:
```
           ID      Name  Age                                              Photo  \
4741   140029  O. Pérez   45  https://cdn.sofifa.org/players/4/19/140029.png

     Nationality                                  Flag  Overall  Potential  \
4741      Mexico  https://cdn.sofifa.org/flags/83.png       71         71

         Club                                        Club Logo  … Composure  \
4741  Pachuca  https://cdn.sofifa.org/teams/2/light/110147.png  …      62.0

     Marking  StandingTackle  SlidingTackle  GKDiving  GKHandling  GKKicking  \
4741    23.0            12.0           11.0      70.0        64.0       65.0

     GKPositioning  GKReflexes Release Clause
4741          73.0        74.0          €272K

[1 rows x 88 columns]
```

[121]:
```
# What is his position?
df[df['Age'] == df['Age'].max()]['Position']
```

[121]:
```
4741    GK
Name: Position, dtype: object
```

[125]:
```
# Who are the 10 fastest players (based on SprintSpeed)?
columns = ['Name', 'Age', 'Nationality', 'Club', 'SprintSpeed']
df.sort_values('SprintSpeed', ascending=False)[:10][columns]
```

[125]:

	Name	Age	Nationality	Club	SprintSpeed
1968	Adama	22	Spain	Wolverhampton Wanderers	96.0
55	L. Sané	22	Germany	Manchester City	96.0
25	K. Mbappé	19	France	Paris Saint-Germain	96.0
1489	I. Bebou	24	Togo	Hannover 96	95.0
36	G. Bale	28	Wales	Real Madrid	95.0
1741	J. Damm	25	Mexico	Tigres U.A.N.L.	95.0
33	P. Aubameyang	29	Gabon	Arsenal	95.0
10928	Maicon	25	Brazil	Livorno	95.0
7204	E. Knudtzon	29	Norway	Lillestrøm SK	94.0
5112	O. Burke	21	Scotland	West Bromwich Albion	94.0

[130]:
```
# Which is the team with the highest number of nationalities?
df.groupby('Club')['Nationality'].nunique().idxmax()
```

[130]:
```
'Brighton & Hove Albion'
```

[134]:
```
# What is the number of players per nationality in Fulham?
df[df['Club'] == 'Fulham'].groupby('Nationality').size()
```

[134]:
```
Nationality
Argentina               1
Australia               1
Belgium                 1
Cameroon                1
DR Congo                1
England                10
France                  2
Germany                 2
Guinea                  1
Ivory Coast             1
Netherlands             1
Norway                  1
Republic of Ireland     1
Scotland                1
Serbia                  1
Spain                   2
Togo                    1
United States           2
Wales                   1
dtype: int64
```

[137]:
```
# How many Hungarian players are there?
(df['Nationality'] == 'Hungary').sum()
```

[137]:
```
38
```

[140]:
```
# Who are the top 5 Hungarian players based on the Overall attribute?
df[df['Nationality'] == 'Hungary'].sort_values('Overall', ascending=False)[:5]
```

[140]:

	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential
512	185122	P. Gulácsi	28	https://cdn.sofifa.org/players/4/19/185122.png	Hungary	https://cdn.sofifa.org/flags/23.png	80	80
614	204638	W. Orban	25	https://cdn.sofifa.org/players/4/19/204638.png	Hungary	https://cdn.sofifa.org/flags/23.png	79	81
1232	230936	A. Nagy	23	https://cdn.sofifa.org/players/4/19/230936.png	Hungary			
1352	182879	B. Dzsudzsák	31	https://cdn.sofifa.org/players/4/19/182879.png	Hungary			
1512	184789	A. Szalai	30	https://cdn.sofifa.org/players/4/19/184789.png	Hungary			

```
1232  https://cdn.sofifa.org/flags/23.png       76         83
1352  https://cdn.sofifa.org/flags/23.png       76         76
1512  https://cdn.sofifa.org/flags/23.png       76         76

                    Club                                        Club Logo  \
512            RB Leipzig  https://cdn.sofifa.org/teams/2/light/112172.png
614            RB Leipzig  https://cdn.sofifa.org/teams/2/light/112172.png
1232              Bologna     https://cdn.sofifa.org/teams/2/light/189.png
1352                  NaN              https://cdn.sofifa.org/flags/23.png
1512  TSG 1899 Hoffenheim   https://cdn.sofifa.org/teams/2/light/10029.png

      … Composure Marking  StandingTackle SlidingTackle  GKDiving  \
512   …      45.0    25.0            16.0           20.0      81.0
614   …      68.0    76.0            83.0           77.0      15.0
1232  …      76.0    75.0            73.0           67.0      10.0
1352  …      77.0    42.0            21.0           20.0      10.0
1512  …      75.0    31.0            41.0           13.0      14.0

      GKHandling  GKKicking GKPositioning GKReflexes Release Clause
512         80.0       76.0          76.0       81.0         €17.9M
614         12.0       15.0          10.0        6.0         €23.1M
1232         7.0        6.0          10.0       14.0         €18.1M
1352         7.0       14.0           7.0       10.0            NaN
1512        12.0        6.0          16.0        9.0         €12.8M

[5 rows x 88 columns]
```

[143]:
```
# Which 10 teams spend the most on wages?
df['Wage']
```

[143]:
```
0        €565K
1        €405K
2        €290K
3        €260K
4        €355K
         …
18202      €1K
18203      €1K
18204      €1K
18205      €1K
18206      €1K
Name: Wage, Length: 18207, dtype: object
```

[153]:
```
def parse_wagestr(s):
    assert s[0] == '€'
    return int(s[1:].replace('K', '000'))
```

```
df['Wage2'] = df['Wage'].apply(parse_wagestr)
```

[160]:
```
df.groupby('Club')['Wage2'].sum().sort_values(ascending=False)[:10]
```

[160]:
```
Club
Real Madrid           5017000
FC Barcelona          4837000
Manchester City       3741000
Manchester United     3391000
Juventus              3292000
Chelsea               3249000
Liverpool             2902000
Tottenham Hotspur     2623000
Arsenal               2588000
FC Bayern München     2286000
Name: Wage2, dtype: int64
```