

Python.py

WARNING! THIS FILE CONTAINS SEVERAL DOCUMENTATIONS AND TEXTS FROM BOOKS THAT I RECOPIED!

MOST OF THE TIME I SIMPLIFIED OR TRANSLATED THEM BUT THEY ARE NOT MY TEXTS!!

there are so many sources and I forgot to quote them but this file should stay for personal use

Eileen Rheinboldt-Tran

2020

**Index**

[Indentation 1](#_Toc72822350)

[Comments 1](#_Toc72822351)

[Variables 1](#_Toc72822352)

[Variable’s names 1](#_Toc72822353)

[Global Keyword 2](#_Toc72822354)

[Operators 2](#_Toc72822355)

[Arithmetic Operators 2](#_Toc72822356)

[Assignment Operators (are used to assign values to variable) 2](#_Toc72822357)

[Comparison Operators (used to compare two values) 3](#_Toc72822358)

[Logical Operators (used to combine conditional statements) 3](#_Toc72822359)

[Identity Operators (used to compare objects, not if they are equal, but if they are the same object, with the same location) 3](#_Toc72822360)

[Membership operators (used to test if a sequence is presented in an object) 3](#_Toc72822361)

[Bitwise Operators (used to compare (binary) numbers) 3](#_Toc72822362)

[print() function 4](#_Toc72822363)

[User input 4](#_Toc72822364)

[Data Types 5](#_Toc72822365)

[Categories of data types 5](#_Toc72822366)

[Getting the Data Type 5](#_Toc72822367)

[Setting the Data Type 5](#_Toc72822368)

[Numbers 6](#_Toc72822369)

[Random number 6](#_Toc72822370)

[Strings 7](#_Toc72822371)

[Multiline Strings 7](#_Toc72822372)

[Slicing 7](#_Toc72822373)

[String length 7](#_Toc72822374)

[Check string 7](#_Toc72822375)

[String format 8](#_Toc72822376)

[Escape character 9](#_Toc72822377)

[String’s Methods 9](#_Toc72822378)

[Booleans 11](#_Toc72822379)

[List 11](#_Toc72822380)

[Access items 11](#_Toc72822381)

[Loop Through a list 12](#_Toc72822382)

[Check if Item 12](#_Toc72822383)

[List Length 12](#_Toc72822384)

[Add Item 12](#_Toc72822385)

[Remove Item 12](#_Toc72822386)

[Copy a list 13](#_Toc72822387)

[List’s Methods 13](#_Toc72822388)

[lists comprehension 13](#_Toc72822389)

[Tuples 14](#_Toc72822390)

[Change Tuple’s values 14](#_Toc72822391)

[Remove/add items 14](#_Toc72822392)

[Tuple’s methods 14](#_Toc72822393)

[Sets 15](#_Toc72822394)

[Access Items 15](#_Toc72822395)

[Add items 15](#_Toc72822396)

[Set length 15](#_Toc72822397)

[Remove items 15](#_Toc72822398)

[Join 2 sets 16](#_Toc72822399)

[Set’s methods 16](#_Toc72822400)

[Dictionaries 17](#_Toc72822401)

[Accessing items 17](#_Toc72822402)

[Change values 17](#_Toc72822403)

[Loop Through a dictionary 17](#_Toc72822404)

[Check if a key exists 18](#_Toc72822405)

[Dictionary length 18](#_Toc72822406)

[Adding items 18](#_Toc72822407)

[Removing items 18](#_Toc72822408)

[Copy a dictionary 19](#_Toc72822409)

[Nested dictionaries 19](#_Toc72822410)

[Dictionary’s methods 19](#_Toc72822411)

[If … Else 20](#_Toc72822412)

[While Loops 20](#_Toc72822413)

[Break statement 20](#_Toc72822414)

[Continue statement 21](#_Toc72822415)

[Else statement 21](#_Toc72822416)

[For Loops 21](#_Toc72822417)

[Looping through a string 21](#_Toc72822418)

[break and continue statement 22](#_Toc72822419)

[range() function 22](#_Toc72822420)

[Else in for loop 22](#_Toc72822421)

[Nested loops 22](#_Toc72822422)

[The pass statement 23](#_Toc72822423)

[Function 23](#_Toc72822424)

[Creating a function 23](#_Toc72822425)

[Calling a function 23](#_Toc72822426)

[Arguments 23](#_Toc72822427)

[Parameters or Arguments? 24](#_Toc72822428)

[Arbitrary Arguments, \*args 24](#_Toc72822429)

[Keyword Arguments (kwargs) 24](#_Toc72822430)

[Arbitrary keyword arguments, \*\*kwargs 25](#_Toc72822431)

[Default parameter value 25](#_Toc72822432)

[Passing a list as an argument 25](#_Toc72822433)

[Return values 25](#_Toc72822434)

[The pass statement 26](#_Toc72822435)

[Recursion 26](#_Toc72822436)

[Lambda 26](#_Toc72822437)

[Why use lambda function? 26](#_Toc72822438)

[Classes and objects 27](#_Toc72822439)

[Create a Class 27](#_Toc72822440)

[Create Object 27](#_Toc72822441)

[The \_\_init\_\_() function 27](#_Toc72822442)

[Object methods 27](#_Toc72822443)

[self parameter 27](#_Toc72822444)

[Modify object properties 28](#_Toc72822445)

[Delete object properties 28](#_Toc72822446)

[The pass statement 28](#_Toc72822447)

[Properties 28](#_Toc72822448)

[Magic or dunder methods 29](#_Toc72822449)

[\_\_dict\_\_ 35](#_Toc72822450)

[Inheritance 35](#_Toc72822451)

[Create a parent class 36](#_Toc72822452)

[Create a child class 36](#_Toc72822453)

[Add the \_\_init\_\_() function 36](#_Toc72822454)

[Use the super() function 36](#_Toc72822455)

[issubclass() function 36](#_Toc72822456)

[The decorators 37](#_Toc72822457)

[Class methods 37](#_Toc72822458)

[Static methods 38](#_Toc72822459)

[Metaclasses 38](#_Toc72822460)

[\_\_new\_\_ method1 38](#_Toc72822461)

[Dynamically create a class 39](#_Toc72822462)

[\_\_new\_\_ method 2 39](#_Toc72822463)

[Iterator 39](#_Toc72822464)

[Iterator vs Iterable 40](#_Toc72822465)

[Looping Through an Iterator 40](#_Toc72822466)

[Create an Iterator 40](#_Toc72822467)

[Modules 41](#_Toc72822468)

[Variables in a module 41](#_Toc72822469)

[Renaming a module 41](#_Toc72822470)

[dir() function 41](#_Toc72822471)

[Import from module 41](#_Toc72822472)

[Dates 42](#_Toc72822473)

[Date output 42](#_Toc72822474)

[Creating Date objects 42](#_Toc72822475)

[The strftime() method 42](#_Toc72822476)

[Time 44](#_Toc72822477)

[Local time function 44](#_Toc72822478)

[Pause the execution of a program 44](#_Toc72822479)

[Strftime method 44](#_Toc72822480)

[Math 44](#_Toc72822481)

[Build-in Math function 45](#_Toc72822482)

[Math Module 45](#_Toc72822483)

[Trigonometry 46](#_Toc72822484)

[fraction module 46](#_Toc72822485)

[JSON 46](#_Toc72822486)

[Parse JSON- Convert from JSON to Python 47](#_Toc72822487)

[Convert from Python to JSON 47](#_Toc72822488)

[Format the result 47](#_Toc72822489)

[Order the result 47](#_Toc72822490)

[RegEx 48](#_Toc72822491)

[RegEx functions 48](#_Toc72822492)

[Metacharacters 48](#_Toc72822493)

[Special sequences 49](#_Toc72822494)

[Sets 49](#_Toc72822495)

[Match object 50](#_Toc72822496)

[Pip 50](#_Toc72822497)

[What is a package? 51](#_Toc72822498)

[Check if Pip is Installed 51](#_Toc72822499)

[Download a package 51](#_Toc72822500)

[Remove a package 51](#_Toc72822501)

[List package 51](#_Toc72822502)

[Try Except 51](#_Toc72822503)

[Exceptions Handling 51](#_Toc72822504)

[Else 52](#_Toc72822505)

[Finally 52](#_Toc72822506)

[Raise an exception 52](#_Toc72822507)

[assert Keyword 53](#_Toc72822508)

[enumerate() function 53](#_Toc72822509)

[File Handling 53](#_Toc72822510)

[Open a file 53](#_Toc72822511)

[Change the current working directory 54](#_Toc72822512)

[Read Files 54](#_Toc72822513)

[Close files 55](#_Toc72822514)

[Delete a file 55](#_Toc72822515)

[Check if file exist 55](#_Toc72822516)

[Delete folder 55](#_Toc72822517)

[Other methods 55](#_Toc72822518)

[With Keyword 56](#_Toc72822519)

[pathlib module 56](#_Toc72822520)

[Save objects into files 56](#_Toc72822521)

[isinstance() function 57](#_Toc72822522)

[sorted() function 57](#_Toc72822523)

[map() function 57](#_Toc72822524)

[Standard flows 58](#_Toc72822525)

[Access standard flows 58](#_Toc72822526)

[Modify standard flows 58](#_Toc72822527)

[Signals 59](#_Toc72822528)

[Intercept a signal 59](#_Toc72822529)

[random module 59](#_Toc72822530)

[Methods 59](#_Toc72822531)

[Password management 61](#_Toc72822532)

[Get a password 61](#_Toc72822533)

[Encrypt a password 61](#_Toc72822534)

[Generate random password 62](#_Toc72822535)

[The network 63](#_Toc72822536)

[TCP protocol (transmission Control Protocol) 63](#_Toc72822537)

[Clients and server 63](#_Toc72822538)

[establish a connection 64](#_Toc72822539)

[Sockets 64](#_Toc72822540)

[The select module 68](#_Toc72822541)

[Unit test with unittest 69](#_Toc72822542)

[Basic structure of a unittest (test case) 70](#_Toc72822543)

[random.choice function test 70](#_Toc72822544)

[Test that an error is raised 71](#_Toc72822545)

[Initialization of tests 71](#_Toc72822546)

[Full recap of test code (with a little bit more) 71](#_Toc72822547)

[mock.patch() 72](#_Toc72822548)

[Assert methods 75](#_Toc72822549)

[Automatic test discovery 76](#_Toc72822550)

[Debug with pdb 77](#_Toc72822551)

[Launch pdb 77](#_Toc72822552)

[Debugger Commands 77](#_Toc72822553)

[Threading 81](#_Toc72822554)

[Functions 81](#_Toc72822555)

[Thread Objects 82](#_Toc72822556)

[Lock objects 82](#_Toc72822557)

[Small example 83](#_Toc72822558)

[\_\_main\_\_ - Top level script environment 84](#_Toc72822559)

[Asynchronous programming with asyncio 84](#_Toc72822560)

[Example of use 85](#_Toc72822561)

[GUI with Tkinter 86](#_Toc72822562)

[Minimum code 86](#_Toc72822563)

[widgets 87](#_Toc72822564)

[Organize our widgets in the window 89](#_Toc72822565)

[commands 90](#_Toc72822566)

[Pause a program 92](#_Toc72822567)

[Virtual environments 92](#_Toc72822568)

[Create a virtual environment 92](#_Toc72822569)

[Activate our virtual environment 92](#_Toc72822570)

[Using requirements file 92](#_Toc72822571)

[Distribute our Python program with cx\_Freeze 93](#_Toc72822572)

[Use the cx\_freeze 93](#_Toc72822573)

[The setup.py file 93](#_Toc72822574)

[keyboard module 94](#_Toc72822575)

[functions 94](#_Toc72822576)

[put a default interpreter on a python script 96](#_Toc72822577)

[Specify the working encoding 96](#_Toc72822578)

[requests module 96](#_Toc72822579)

[Methods 97](#_Toc72822580)

[requests response object 97](#_Toc72822581)

[Creating Web APIs with Flask 99](#_Toc72822582)

[minimal Flask App 99](#_Toc72822583)

[variable rules 99](#_Toc72822584)

[Redirection behavior 100](#_Toc72822585)

[returns status code 100](#_Toc72822586)

[Before request 100](#_Toc72822587)

[app.run() parameter 101](#_Toc72822588)

[Blueprints 101](#_Toc72822589)

[logging 102](#_Toc72822590)

[query string 102](#_Toc72822591)

[NumPy 103](#_Toc72822592)

[The basics 103](#_Toc72822593)

[Array creation 104](#_Toc72822594)

[printing array 106](#_Toc72822595)

[Adding, removing, and sorting elements 106](#_Toc72822596)

[Basic operation 107](#_Toc72822597)

[Universal functions 108](#_Toc72822598)

# Indentation

Indentation refers to the spaces at the beginning of a code line.

⚠In Python it’s very important

It indicates a block of code.

We have to use the same number of spaces in the same block of code.

# **Comments**

Comment starts with a # and Python will ignore them.

# Variables

Variables are containers for storing data values.

A variable is created the moment we first assign a value to it.

⚠We cannot combine strings and numbers



## Variable’s names

**Rules:**

* A variable name must start with a letter or an underscore character (\_)
* A variable name cannot start with a number
* A variable name can only contain: A-z, 0-9 and \_
* Variables’ names are case-sensitive (age ≠ Age ≠ AGE)

## Global Keyword

Global variables can be used by everyone, both inside and outside the function.

Variables who aren’t in function are automatically global.

When we create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, we can use the global keyword.



# Operators

## Arithmetic Operators

+ Addition - Subtraction \* Multiplication /division % Modulus \*\* Exponentiation // Floor division

## Assignment Operators (are used to assign values to variable)

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same as |
| = | x=5 | x=5 |
| += | x+=3 | x=x+3 |
| -= | x-=3 | x=x-3 |
| \*= | x\*=3 | x=x\*3 |
| /= | x/=3 | x=x/3 |
| %= | x%=3 | x=x%3 |
| //= | x//=3 | x=x//3 |
| \*\*= | x\*\*=3 | x=x\*\*3 |
| &= | x&=3 | x=x&3 |
| |= | x|=3 | x=x|3 |
| ^= | X^=3 | x=x^3 |
| >>= | x>>=3 | x=x>>3 |
| <<= | X<<=3 | x=x<<3 |

## Comparison Operators (used to compare two values)

== Equal != Not Equal > Greater Than < Less Than >= Greater than or equal to <= Less Than or equal to

## Logical Operators (used to combine conditional statements)

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| and | Return True if both statements are True | x<5 and x<10 |
| or | Returns True if one of the statements is True | x<5 or x<4 |
| not | Reverse the result, returns False if the result is True | not(x<5 and x<10) |

## Identity Operators (used to compare objects, not if they are equal, but if they are the same object, with the same location)

is Returns True if both variables are the same object ⚠is ≠ ==

is not Returns True if the variables are not the same object ⚠is not ≠ !=

## Membership operators (used to test if a sequence is presented in an object)

in Returns True if a sequence with the specified value is present in the object

not in Returns True if a sequence with the specified value is not present in the object

## Bitwise Operators (used to compare (binary) numbers)

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| & | AND | Sets each bit to 1 if both bits are1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left and let the rightmost bits fall off |

# print() function

print the specified message to the screen, or other standard output device.

The message can be a string, or any other objects, the object will be converted into a string before written to the screen.

**Parameter=**

* Object(s) Any object, and as many as we like
* sep=” separator” Optional. Specify how to separate objects, if there is more than Default is “ “.
* end= ”end” Optional. Specify what to print at the end. Default is “\n”
* file Optional. An object with a write method. Default is sys.stdout
* flush Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

# User input

Python allows for user input.

That means we are able to ask the user for input.



# Data Types

## Categories of data types

* Text type: str
* Numeric types: int, float, complex
* Sequence types: list, tuple, range
* Mapping type: dict
* Set types: set, frozenset
* Boolean type: bool
* Binary types: bytes, bytearray, memoryview

## Getting the Data Type

To get the data type of any object we use the type() function.

## Setting the Data Type

|  |  |  |
| --- | --- | --- |
| Data Type | Setting the data type (example) | Setting the specific data type (example) |
| str | x=”Hello World” | x=str(“Hello World”) |
| int | x=20 | x=int(20) |
| float | x=20.5 | x=float(20.5) |
| complex | x=1j | x=complex(1j) |
| list | x=[“apple”,”banana”,”cherry”] | x=list((“apple”,”banana”,”cherry »)) |
| tuple | x=(“apple”,”banana”,”cherry”) | x=tuple((“apple”,”banana”,”cherry“)) |
| range | x=range(6) | x=range(6) |
| dict | x={“name”:”John”,”age”:36} | x=dict(name= “John”, age=36) |
| set | x={“apple”,”banana”,”cherry”} | x=set((“apple”,”banana”,”cherry”)) |
| frozenset | x=frozenset({“apple”,”banana”, ”cherry”}) | x=frozenset((“apple”,”banana”, ”cherry”)) |
| bool | x=True | x=bool(5) |
| bytes | x=b”Hello” | x=bytes(5) |
| bytearray | x=bytearray(5) | x=bytearray(5) |
| memoryview | x=memoryview(bytes(5)) | x=memoryview(bytes(5)) |

# Numbers

There are 3 numeric types in Python:

* int integer is a whole number, positive or negative, without decimals
* float floating point number is a number, positive or negative containing decimals
* complex complex numbers are written with a “j” as the imaginary part

Type Conversion= We can convert from one type to another with int(), float() , and complex() methods.

## Random number

Python does not have a random() function to make a random number, but has a built-in module called random.



# Strings

String literals are surrounded by either single or double quotations mark.

## Multiline Strings

We can assign a multiline string to a variable by using 3 single/double quotes



⚠Strings are arrays

## Slicing

We can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string



We use negative indexes to start the slice from the end of the string.

## String length

To get the length of a string, we use the len() function.

## Check string

To check if a certain phrase or character is present in a string, we can use the keywords in and not in.



## String format

We can’t combine strings and number with +.

So, there are 2 methods:

### 1)format() method

The format() method formats the specified value(s) and insert them inside the string's placeholder.

The placeholder is defined using curly brackets: {}.

The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.

Inside the placeholders you can add a formatting type to format the result:

|  |  |
| --- | --- |
| :< | Left aligns the result (within the available space) |
| :> | Right aligns the result (within the available space) |
| :^ | Center aligns the result (within the available space) |
| := | Places the sign to the left most position |
| :+ | Use a plus sign to indicate if the result is positive or negative |
| :- | Use a minus sign for negative values only |
| : | Use a space to insert an extra space before positive numbers (and a minus sign before negative numbers) |
| :, | Use a comma as a thousand separator |
| :\_ | Use a underscore as a thousand separator |
| :b | Binary format |
| :c | Converts the value into the corresponding unicode character |
| :d | Decimal format |
| :e | Scientific format, with a lower-case e |
| :E | Scientific format, with an upper-case E |
| :f | Fix point number format |
| :F | Fix point number format, in uppercase format (show inf and nan as INF and NAN) |
| :g | General format |
| :G | General format (using a upper-case E for scientific notations) |
| :o | Octal format |
| :x | Hex format, lower case |
| :X | Hex format, upper case |
| :n | Number format |
| :% | Percentage format |

### 2) f-string

f-strings are string literals that have an f at the beginning and curly braces containing expression that will be replaced with their values.

The syntax is similar to the one we used with str.format() but less verbose

## Escape character

⚠If we want to have a string that contain a “\” but we don’t want it to be treated as an escape character, we use the raw string r” ”

To insert characters that are illegal in a string, we use “\” followed by the character we want to insert).

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Result | Code | Result |
| \” | Double quote | \b | Backspace |
| \\ | Backslash | \f | Form feed |
| \n | New line | \ooo | Octal value |
| \r | Carriage return | \xhh | Hex value |
| \t | Tab |  |  |

## String’s Methods

⚠All string methods return new values. They do not change the original string

|  |  |
| --- | --- |
| Method | Description |
| capitalize() | Converts the first character to upper case |
| casefold() | Converts string into lower case |
| center() | Returns a centered string |
| count() | Returns the number of times a specified value occurs in a string |
| encode() | Returns an encoded version of the string |
| endswith() | Returns true if the string ends with the specified value |
| expandtabs() | Sets the tab size of the string |
| find() | Searches the string for a specified value and returns the position of where it was found |
| format() | See *format() method* |
| index() | Searches the string for a specified value and returns the position of where it was found |
| isalnum() | Returns True if all characters in the string are alphanumeric |
| isalpha() | Returns True if all characters in the string are in the alphabet |
| isdecimal() | Returns True if all characters in the string are decimals |
| isdigit() | Returns True if all characters in the string are digits |
| isidentifier() | Returns True if the string is an identifier |
| islower() | Returns True if all characters in the string are lower case |
| isnumeric() | Returns True if all characters in the string are numeric |
| isprintable() | Returns True if all characters in the string are printable |
| isspace() | Returns True if all characters in the string are whitespaces |
| istitle() | Returns True if the string follows the rules of a title |
| isupper() | Returns True if all characters in the string are upper case |
| join() | Joins the elements of an iterable to the end of the string |
| ljust() | Returns a left justified version of the string |
| lower() | Converts a string into lower case |
| lstrip() | Returns a left trim version of the string |
| maketrans() | Returns a translation table to be used in translations |
| partition() | Returns a tuple where the string is parted into three parts |
| replace() | Returns a string where a specified value is replaced with a specified value |
| rfind()=rindex() | Searches the string for a specified value and returns the last position of where it was found |
| rjust() | Returns a right justified version of the string |
| rpartition() | Returns a tuple where the string is parted into three parts |
| rsplit() | Splits the string at the specified separator, and returns a list |
| rstrip() | Returns a right trim version of the string |
| split() | Splits the string at the specified separator, and returns a list |
| splitlines() | Splits the string at line breaks and returns a list |
| startswith() | Returns true if the string starts with the specified value |
| strip() | Returns a trimmed version of the string |
| swapcase() | Swaps cases, lower case becomes upper case and vice versa |
| title() | Converts the first character of each word to upper case |
| translate() | Returns a translated string |
| upper() | Converts a string into upper case |
| zfill() | Fills the string with a specified number of 0 values at the beginning |

# Booleans

Booleans represent one of two values: True or False

The bool() function allows us to evaluate any values, and return True or False.

# List

A list is a collection which is ordered and changeable and indexed. It’s written with square brackets.



## Access items

We can access the list items by referring to the index number. Negative index accesses the list from the end.

Range of indexes= We can specify a range of indexes by specifying where to start and where to end the range. It’ll return a new list with the specifies items.



## Loop Through a list

We can loop through the list items by using a for loop:



## Check if Item

To determinate if a specified item is present in a list, we use the in keyword:



## List Length

To determinate how many items a list has, use the len() function.

## Add Item

To add an item to the end of a list, we use the append() method.

To add an item at the specified index, we use the insert method.



## Remove Item

The remove() method removes the specified item.

The pop() method removes the specified index, or the last item if no index specified

The del keyword removes the specified index or the list completely

The clear() method empties the list.

## Copy a list

We can’t copy a list simply by typing list2=list1, because every change in list1 will affect list2.

There are 2 ways=

1. copy() x=thislist.copy()
2. list() x=list(thislist)

## List’s Methods

|  |  |
| --- | --- |
| Method | Description |
| append() | See *Add Item* |
| clear() | See *Remove Item* |
| copy() | See Copy a list |
| count() | Returns the number of elements with the specified value |
| sum() | Calculate the total of a list |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |
| insert() | See *Add Item* |
| pop() | See *Remove Item* |
| remove() | See *Remove Item* |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |
| max() | Find the biggest number |
| min() | Find the smallest number |

## lists comprehension

Syntax: new\_sequence=[element for element in old\_sequence if condition]



# Tuples

A tuple is a collection which is ordered, unchangeable and indexed. It’s written with round brackets. 

To create a tuple with one item, we have to put a comma after the item, otherwise Python will not recognize it’s a tuple. 

## Change Tuple’s values

Once a tuple is created, we can’t change its values. But there is a workaround. We can convert a tuple into a list, change the list, and convert the list back into a tuple.



## Remove/add items

⚠Tuples are unchangeable, we cannot add or remove items!!!

But we can delete the tuple completely with the del keyword.

## Tuple’s methods

count() Returns the number of times a specified value occurs in a tuple

index() Searches the tuple for a specified value and return the position of where it was found.

# Sets

A set is a collection which is unordered, unchangeable and unindexed. It’s written with curly brackets.

⚠We can’t change items but we can add or remove some

## Access Items

We can’t access an item by an index because sets are unordered.

But we can loop through the set’s items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

## Add items

To one item to a set, we use the add() method.

To add more than one item, we use the update() method.



## Set length

To determinate how many items a set has, we use the len() function.

## Remove items

To remove an item in a set, we use the remove(), or the discard() methods.

The difference between these two methods are if the item to remove does not exist remove() will raise an error and discard() will not raise an error.

We can also use the pop() method but we will not know which item has been removed.

The clear() method empties the set.

The del keyword will delete the set completely.

## Join 2 sets

We can use the union() method (|) that return a new set containing all items from both sets:



Or the update() method that insert all the items from one set into another.

⚠Both union() and update() will exclude any duplicate items.

## Set’s methods

|  |  |
| --- | --- |
| Method | Description |
| add() | See *Add items* |
| clear() | See *Remove items* |
| copy() | Returns a copy of the set |
| difference() / (-) | Returns a set containing the difference between two or more sets |
| difference\_update() | Removes the items in this set that are also included in another, specified set |
| discard() | See *Remove items* |
| intersection() / (&) | Returns a set, that is the intersection of two other sets |
| intersection\_update() | Removes the items in this set that are not present in other, specified set(s) |
| isdisjoint() | Returns whether two sets have an intersection or not |
| issubset() / (<=) | Returns whether another set contains this set or not |
| issuperset() / (>=) | Returns whether this set contains another set or not |
| pop() | See *Remove items* |
| remove() | See *Remove items* |
| symmetric\_difference() | Returns a set with the symmetric differences of two sets |
| symmetric\_difference\_update() | inserts the symmetric differences from this set and another |
| union() | See *Join 2 sets* |
| update() | See *Add items* |

# Dictionaries

A dictionary is a collection which is unordered, changeable and indexes. They are written with curly brackets, and they have keys and values.

## Accessing items

We can access an item of a dictionary by referring to its key name, inside square brackets. 

The method get() give the same result. 

## Change values

We can change the value of a specific item by referring to its key name.

## Loop Through a dictionary

We can loop through a dictionary by using a for loop.

When looping through a dictionary, the return values are the keys of the dictionary, but there are methods to return the values as well.

Print all the key’s name in the dictionary, one by one:



Print all the values in the dictionary, one by one:



or we can use the values() method:

Loop through both keys and values, by using the items() method:



## Check if a key exists

To determinate if a specified key is present in a dictionary, we use the in keyword.

## Dictionary length

To determinate how many items a dictionary has, we use the len() function.

## Adding items

Adding an item to the dictionary is done by using a new index key and assigning a value to it. 

## Removing items

The pop() method removes the item with the specified key name.

The popitem() method removes the last inserted item.

The del keyword removes the item with the specified key name or delete the dictionary completely.

The clear() method empties the dictionary.

## Copy a dictionary

We can’t copy a dictionary by using dict2=dict1, because every change in dict1 will affect dict2.

There are 2 ways=

1. copy() mydict=thisdict.copy()
2. dict() mydict=dict(thisdict)

## Nested dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.  

## Dictionary’s methods

|  |  |
| --- | --- |
| Method | Description |
| clear() | See *Removing items* |
| copy() | See *Copy a dictionary* |
| fromkeys() | Returns a dictionary with the specified keys and value |
| get() | See *Accessing items* |
| items() | See Loop Through a dictionary |
| keys() | Returns a list containing the dictionary's keys |
| pop() | See *Removing items* |
| popitem() | See *Removing items* |
| setdefault() | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| update() | Updates the dictionary with the specified key-value pairs |
| values() | See *Loop Through a dictionary* |

# If … Else

if= a way of saying “if True …”

elif= a way of saying “if the previous condition were not True, then try this condition”.

Short Hand if= 

Short Hand If…Else= 

We can also have multiple else:

or/and= combine conditional statements 

nested if= we can have if statements inside if statements

pass statement= if statement can’t be empty, but if we need an if statement with no content, we put a pass statement to avoid an error.

# While Loops

Execute a set of statements as long as a condition is True.

## Break statement

With the break statement we can stop the loop.



## Continue statement

With the continue statement we can stop the current iteration, and continue with the next one.



## Else statement

With the else statement we can run a block of code once, when the condition is no longer True.



# For Loops

Is used for iterating over a sequence (list / tuple / dictionary / set / string).

This is less like the for keyword in other programming languages, and work more like an iterator method as found in other object-oriented programming languages.

## Looping through a string



## break and continue statement

With the break statement we can stop the loop before it has looped through all the items.

With the continue statement we can stop the current iteration of the loop, and continue with the next one.

## range() function

Th range() function returns a sequence of numbers, starting from 0 (by default), and increment by 1 (by default), and ends at a specified number.



## Else in for loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

## Nested loops

A nested loop is a loop inside a loop.



## The pass statement

For loops can’t be empty, but if we for some reason have a for loop with no content, we put in the pass statement to avoid getting an error.

# Function

A function is a block of code which only runs when it is called.

We can pass data, known as parameter, into a function.

A function can return a data as result.

## Creating a function

In python, a function is define using the def keyword.

## Calling a function

To call a function, we use the function’s name followed by parentheses.

## Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function’s name, inside parentheses. We can add as many arguments as we want , just we separate them with a comma.

By default, a function must be called with the correct number of arguments. Meaning that if a function expects 2 arguments, we have to call the function with 2 arguments, not more, and not less.

## Parameters or Arguments?

The terms parameter and argument can be used for the same thing: information that are passed into a function.

From a function perspective:

* A parameter is the variable inside the parentheses in the function definition.
* An argument is the value that is sent to the function when it’s called.

## Arbitrary Arguments, \*args

If we don’t know how many arguments will be passed into our function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly. 

## Keyword Arguments (kwargs)

We can also send arguments with the key=value syntax.

This way the order of the arguments does not matter.



## Arbitrary keyword arguments, \*\*kwargs

If we do not know how many arguments will be passed into our function, we add two asterisks: \*\* before the parameter’s name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly.



## Default parameter value

If we call the function without argument, it uses the default value.

## Passing a list as an argument

We can send any data types of arguments to a function (string, number ,list, dictionary, …), and it will be treated as the same data type inside the function.

## Return values

To let a function return a value, we use the return statement.



## The pass statement

Function definitions can’t be empty, but if for some reason we have a function definition with no content, put in the pass statement to avoid getting an error.

## Recursion

Python accepts function recursion, which means a defined function can call itself.



# Lambda

A lambda function is a small anonymous function.

It can take any number of arguments, but can only have one expression.

Syntax= lambda arguments: expression



## Why use lambda function?

The power of lambda is better shown when we use them as an anonymous function inside another function.

Say we have a function that takes one argument, and that argument will be multiplied with an unknown number:



# Classes and objects

## Create a Class

To create a class, we use the keyword class.

## Create Object



## The \_\_init\_\_() function

The example above are classes and object in their simplest form, and are not really useful in real life application.

All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.

We use the \_\_init\_\_() function to assign values to object properties, or other operation that are necessary to do when the object is being created.



## Object methods

Methods in objects are functions that belong to the object.

## self parameter

The self parameter is a reference to the current instance of the class, and is used to access variable that belongs to the class.

⚠It doesn’t have to be named self, we can call it whatever we like, but it has to be the first parameter of any function in the class.

## Modify object properties



## Delete object properties

We can delete properties on objects by using the del keyword. 

## The pass statement

Class definition can’t be empty, but if we have a class definition with no content, we put in the pass statement to avoid getting an error.

## Properties

Properties are used to control access to certain attributes of an instance.

They are defined in the class corp.

Read properties are methods topped with the @property line.

Write properties must be preceded by an @property\_name.setter line.

The convention used for private attributes is to prefix their name with an underscore (\_).



## Magic or dunder methods

Dunder methods allow to influence the way Python accesses attributes of an instance and reacts to certain operators or conversions.

The special methods are all surrounded by two signs “underscore” (\_).

### Editing the object

#### \_\_init\_\_

Controls the creation of our attributes.

See *The \_\_init\_\_() function*

#### \_\_del\_\_

### Object representation

Two special methods control how the object is represented and displayed.

#### \_\_repr\_\_

It affects the way the object is displayed when we type its nae directly.



#### \_\_str\_\_

Specially used to display the object with print.

If no \_\_str\_\_ method is defined, Python calls the object’s \_\_repr\_\_ method.



### Access the attributes of our object

A small function: hasattr(object, “name”) #return True if the attr “name” exist, False if not

#### \_\_getattr\_\_

This method is called when we type object.attribute, and if it doesn’t find the attribute in the object and if a \_\_getattr\_\_ exists, it calls it by passing it as a parameter the name of the attribute sought.



#### \_\_setattr\_\_

This method defines access to an attribute intended to be modified. If we write object.attribute = new\_value, the magic \_\_setattr\_\_ method will be called as follow: \_\_setattr\_\_(“attribute\_name”, new\_value)



#### \_\_delattr\_\_

This method is called when we want to delete an attribute from the object by writing del object.attribute for example.



### Access to the element of a container

#### \_\_getitem\_\_, \_\_setitem\_\_, \_\_delitem\_\_

These methods are used respectively to define what to do when writing=

* object[index]
* object[index]=value
* del object[index]



#### \_\_contains\_\_ (method behind the keyword in)



This method is used when we want to know if an object is a container



#### \_\_len\_\_

Allows to know the size of a container.



### Mathematical methods

We will use this for all the examples:

#### \_\_add\_\_ (operator overload +)



It takes as parameter the object that we want to add.



**On the same model they are the methods:**

#### \_\_sub\_\_ (operator overload -)

#### \_\_mul\_\_ (operator overload \*)

#### \_\_trudiv\_\_ (operator overload /)

#### \_\_floordiv\_\_ (operator overload //)

#### \_\_mod\_\_ (operator overload %)

#### \_\_pow\_\_ (operator overload \*\*)

* **…**

We can do d1+4 but we can’t do 4+d1. If we want to do it, we just prefix the name of the special methods with an r(right)

To overload operators such as +=, -=, … we prefix the name of the methods with i (increment)

**Magic methods to overload comparison operators:**

#### \_\_eq\_\_ (operator overload ==)

#### \_\_ne\_\_ (operator overload !=)

#### \_\_gt\_\_ (operator overload >)

#### \_\_ge\_\_ (operator overload >=)

#### \_\_lt\_\_ (operator overload <)

#### \_\_le\_\_ (operator overload <=)

Note= If Python fails to evaluate object1<object2, it will try the opposite operation, i.e., object2>=object1.

### Magic method useful to pickle

#### \_\_getstate\_\_

This method is called when serializing the object. \_\_getstate\_\_ is called just before recording.

If no \_\_getstate\_\_ method is defined, pickle stores the dictionary of attributes of the object to save (it’s contained in object.\_\_dict\_\_)

#### \_\_setstate\_\_

The same as \_\_getstate\_\_ but is called after deserialization

## \_\_dict\_\_

The magic \_\_dict\_\_ attribute is a dictionary that contains the names of attributes associated with their values.



# Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherit from another class, also called derived class.

## Create a parent class

Any class can be a parent class, so the syntax is the same as creating any other class.

## Create a child class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.

Note= Use the pass keyword when we don’t want to add any other properties or methods to the class.

## Add the \_\_init\_\_() function

When we add the \_\_init\_\_() function, the child class will no longer inherit the parent’s \_\_int\_\_() function.

Note= The child’s \_\_init\_\_() overrides the inheritance of the parent’s \_\_init\_\_() function.

To keep the inheritance of the parent’s \_\_init\_\_() function, we add a call to the parent’s \_\_init\_\_() function:



## Use the super() function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent:



# issubclass() function

It returns True if the specified object is a subclass of the specified object, otherwise False.

Syntax : issubclass(object,subclass)



# The decorators

Decorators are a way to modify the “default” behavior of functions.

This is an example of metaprogramming which is the writing of programs that manipulate.



## Class methods

They don’t want work on the instance (self), but on the class itself (cls)

## Static methods

They don’t take an instance(self), nor the class(cls). They are function but which are defined in the body of the class to regroup together functionality.



# Metaclasses

The concept is to generate classes from other classes.

## \_\_new\_\_ method1

\_\_init\_\_ method initiates the object and \_\_new\_\_ create it.



⚠Classes being objects they are all modeled on a metaclass which is, unless explicitly modified, type.

## Dynamically create a class



## \_\_new\_\_ method 2



# Iterator

An iterator is an object that contain a countable number of values.

It’s an object that can be iterated upon, meaning that we can traverse through all the values.

It’s an object which implements the iterator protocol, which consist of the methods \_\_iter\_\_() and \_\_next\_\_().

## Iterator vs Iterable

Strings, lists, tuples, dictionaries, and sets are all iterable objects. They are iterable *containers* which we can get an iterator from.

All these objects have a iter() method which is used to get an iterator.



## Looping Through an Iterator

We can also use a for loop to iterate through an iterable object.

## Create an Iterator

To create an object/class as an iterator we have to implement the methods \_\_iter\_\_() and \_\_next\_\_() to our object.

To prevent the iteration to go on forever, we can use the StopIteration statement.



# Modules

To create a module, we just save the code we want in a file with the file extension .py

Now we can import the module we just create, by using the import statement.

## Variables in a module

The module can contain functions, but also variables of all types( dictionaries,object, …)



## Renaming a module

We can create an alias when we import a module, by using the as keyword.



## dir() function

The dir() function is a built-in function to list all the function’s names (or variable’s name) in a module.

## Import from module

We can choose to import only parts from a module, by using the from keyword



⚠When importing using the from keyword, don’t use the module name when referring to element in module.



# Dates

A date in Python is not a data type of its own, but we can import the module named datetime to work with dates as date object.

## Date output

The datetime module has many methods to return information about the date object. Here a few examples:



## Creating Date objects

To create a date, we can use the datetime() class (constructor) of the datetime module.

It requires 3 parameters to create a date : year/month/day.



The datetime() class also takes parameter for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and as a default value of 0 (none for timezone).

## The strftime() method

The datetime object has a method for formatting date objects into readable strings.

This method is called strftime(), and takes one parameter, format, to specify the format of the returned string.



|  |  |  |
| --- | --- | --- |
| Directive | Description | Example |
| %a | Weekday, short version | Wed |
| %A | Weekday, full version | Wednesday |
| %w | Weekday as a number 0-6, 0 is Sunday | 3 |
| %d | Day of the month 01-31 | 31 |
| %b | Month name, short version | Dec |
| %B | Month name, full version | December |
| %m | Month as a number 01-12 | 12 |
| %y | Year, short version, without century | 21 |
| %Y | Year, full version | 2021 |
| %H | Hour 00-23 | 17 |
| %I | Hour 00-12 | 5 |
| %p | AM/PM | PM |
| %M | Minute 00-59 | 41 |
| %S | Seconds 00-59 | 08 |
| %f | Microseconds 000000-999999 | 548513 |
| %z | UTC offset | +0100 |
| %Z | Timezone | CST |
| %j | Day number of a year 001-366 | 365 |
| %U | Week number of a year, Sunday as the first day of the week, 00-53 | 52 |
| %W | Week number of a year, Monday as the first day of the week, 00-53 | 52 |
| %c | Local version of date and time | Mon Dec 31 17:41:00 2020 |
| %x | Local version of date | 12/31/2020 |
| %X | Local version of time | 17:41:00 |
| %% | A % character | % |

# Time

There is a module named time.

## Local time function

Returns an object containing, in order:

1. tm\_year= year as an integer
2. tm\_mon= number of the month(1-12)
3. tm\_day= number of the day(1-31)
4. tm\_hour= hour of the day (1-31)
5. tm\_min= number of minutes(0-59)
6. tm\_sec= number of seconds(0-61 or 0-59)
7. tm\_wday= day of the week as an integer(0-6, 0 is Monday)
8. tm\_yday= day of the year (1-366)
9. tm\_isdst= integer representing the local time change

The localtime function can take one parameter: the timestamp, if the parameter is not specified, the function will automatically use the time.time() and will therefore return the current date and time.

## Pause the execution of a program

The function is called sleep and takes as parameter an integer or a float.



## Strftime method

See *The strftime() method~*

# Math

Python has a set of build-in math function, including an extensive math module, that allows us to perform mathematical task on numbers.

## Build-in Math function

The min() and max() functions can be used to find the lowest or highest value in an iterable:



The abs() function returns the absolute (positive) value of the specified number:



The pow(x,y) function returns the value of x power of y:



## Math Module

Python has also a built-in module called math, to use it, we must import it.

**Some methods (not all):**

The math.sqrt() method returns the square root of a number:



The math.ceil() method rounds a number upwards to its nearest integer, and the math.floor() method rounds a number downwards to its nearest integer, and returns the result:



The math.pi constant returns the value of PI:



## Trigonometry

⚠In Python angles are given and returned in radians (rad)

**Math function that convert angles:**

* math.degrees(angle\_in\_radians) #convert into degrees
* math.radians(angle\_in\_degrees) #convert into radians

**Trigonometric functions:**

* cos: cosines
* sin: sinus
* tan: tangent
* acos: arc cosine
* asin: arc sinus
* atan: arc tangent

# fraction module

What interest us=



To create a fraction from a float we use the from\_float class method.



# JSON

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notations.

Python has a built-in package called json, which can be used to work with JSON data.

## Parse JSON- Convert from JSON to Python

If we have a JSON string, we can parse it by using the json.loads() method.

⚠The result will be a Python dictionary



## Convert from Python to JSON

If we have a Python object, we can convert it into JSON string by using json.dumps() method.



When we convert from Python to JSON, Python objects are converted into JSON (JavaScript) equivalent.

## Format the result

The json.dumps() method has parameters to make it easier to read the result.

Use the indent parameter to define the number of indents: 

We can also define the separator, default value is (“ , “, “: ”)), which means using a comma and a space to separate each object, and a colon and a space to separate keys from value :

## Order the result

The json.dumps() method has parameter to order the key in the result.

Use the sort\_keys parameter to specify if the result should be sorted or not:



# RegEx

A RegEx, or Regular Expression, is a sequence of characters that form a search pattern.

Python has a built-in package called re, which can be used to work with RegEx.

## RegEx functions

The re module offers a set of functions thar allows us to search a string for a match:

* findall() Returns a list containing all matches
* search() Returns a Match object if there is a match anywhere in the string
* split() Returns a list where the string has been split at each match
* sub() Replaces one or many matches with a string.



## Metacharacters

Metacharacters are character with a special meaning:

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| [ ] | A set of characters | “[a-m]” |
| \ | Signal a special sequence (can also be used to escape special character) | “\d” |
| . | Any character (except newline character) | “he..o” |
| ^ | Starts with | “ˆhello” |
| $ | Ends with | “world$” |
| \* | Zero or more occurrence | “aix\*” |
| + | One or more occurrence | “aix+” |
| ? | Zero or one occurrence | “aix?” |
| { } | Exactly the specified number of occurrences | “al{2}” |
| ¦ | Either or | “falls¦stays” |
| ( ) | Capture and group |  |

## Special sequences

A special sequence is a \ followed by one of the list below, and has a special meaning:

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| \A | Returns a match if the specified characters are at the beginning of the string | “\Athe” |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word (The “r” in the beginning is making sure that the string is being treated as a “raw string”) | r”\bain” |
| \B | Returns a match where the specified characters are present, but NOT at the beginning or at the end of a word. | r”\Bain”  r”ain\B” |
| \d | Returns a match where the string contains digit (number from 0-9) | “\d” |
| \D | Returns a match where the string DOES NOT contain digit | “\D” |
| \s | Returns a match where the string contains a white space character | “\s” |
| \S | Returns a match where the string DOES NOT contain a white space character | “\S” |
| \w | Returns a match where the string contains any word character (character from a-Z, digit from 0-9, and the underscore \_ character) | “\w” |
| \W | Returns a match where the string DOES NOT contain any word character | “\W” |
| \Z | Returns a match if the specified characters are at the end of a string | “Spain\Z” |

## Sets

A set is a set of characters inside a pair of square brackets [ ] with a special meaning:

|  |  |
| --- | --- |
| Set | Description |
| [arn] | Returns a match where one of the specified characters (a, r, or n) are present. |
| [a-n] | Returns a match for any lower-case character, alphabetically between a and n. |
| [^arn] | Returns a match for any characters EXCEPT a, r, and n |
| [0123] | Returns a match where any of the specified digits (0,1,2,3) are present. |
| [0-9] | Returns a match for any digit between 0 and 9 |
| [0-5][0-9] | Returns a match for any two-digit numbers from 00 and 59. |
| [a-zA-Z] | Returns a match for any character alphabetically between a and z, lower case OR upper case. |
| [+] | In sets, +, \*, ., ¦, ( ), $, { } has no special meaning so [+] means:  Return a match for any + character in the string. |

## Match object

A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match object.

The Match Object has properties and methods used to retrieve information about the search, and the result:

* span() returns a tuple containing the start - , and end position of the match.



* string returns the string passed into the function



* group() returns the part of string where there was a match



# Pip

See Virtual environments or Using requirements file

Pip is a package manager for Python packages, or modules.

## What is a package?

A package contains all the files we need for a module.

Modules are Python code libraries we can include in our project.

## Check if Pip is Installed

Navigate our command line to the location of Python’s script directory, and type:

pip --version

## Download a package

Navigate our command line to the location of Python’s script directory, and type:

pip install name\_of\_the\_package

To use a package, we import it into our project.

## Remove a package

pip uninstall name\_of\_the\_package

## List package

Pip list

# Try Except

The try block lets us text a block of code for error.

The except block lets us handle the error.

The finally block lets us execute code, regardless of the result of the try and except blocks.

## Exceptions Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:



Since the try block raise an error, the except block will be executed.

## Else

We can use the else keyword to define a block of code to be executed if no errors where raised.

## Finally

The finally block, if specified, will be executed regardless if the try block raise an error or not.

This can be useful to close objects and clean up resources:



## Raise an exception

We can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, we use the raise keyword.



We can define what kind of error to raise, and the text to print to the user.



# assert Keyword

It’s used when debugging code.

It lets us test if a condition in our code is True, if not, the program will raise an AssertionError.

We can write a message to be written if the code return False.



# enumerate() function

The enumerate() function takes a collection and returns it as an enumerate object.

It adds a counter as the key of the enumerate object.

Syntax: enumerate(iterable, start)



# File Handling

## Open a file

To open a file, we use the open() function. It takes 2 parameters : filename, and mode.

There are 4 different methods (modes) for opening a file:

* “r” = Read – Default value. Opens a file for reading, error if the file does not exist.
* “a” = Append – Opens a file for appending, creates the file if it doesn’t exist.
* “w” = Write – Opens a file for writing, creates the file if it doesn’t exist.
* “x” = Create – Creates the specified file, returns an error if the file exists.

In addition, we can specify if the file should be handled as binary or text mode.

* “t” = Text – Default value. Text mode
* “b” = Binary – Binary mode (ex= images)

Syntax:



## Change the current working directory

To change the current working directory, we use a function from the os module, chdir (change directory).



⚠ We use / or \\ not just one \ otherwise there will be an error.

Or we can directly write the location: 

⚠ The things are that we have to rewrite the location each time we reuse the variable to avoid error.

## Read Files

By default, the read() method returns the whole text, but we can also specify how many characters we want to return. 

We can return one line by using the readline() method:



By looping through the lines of the file, we can read the whole file, line by line.

## Close files

We should always close our files when we are done with it, because in some case, due to buffering, changes made to a file may not show until the file is close.



⚠ To verify if a file is closed, we can type = filename.closed (return True or False)

## Delete a file

To delete a file, we must import the os module, and run its os.remove() function.



## Check if file exist

To avoid getting an error, we might want to check if the file exists before we try to delete it.



## Delete folder

To delete an entire Folder, we use the os.rmdir() method.



⚠We can only remove EMPTY folders!!!

## Other methods

readable() = returns True if the file is readable, False if not.

readlines() = returns a list containing each line in the file as a list item. Use the hint parameter to limit the number of lines returned, the default is -1, which mean all lines will be returned.

writable() = returns True or False if the file is writable or not.

writelines() = writes the items of a list to the file.

## With Keyword

Allows that if an exception occurs the file is closed anyway at the end of the block.

Syntax:



## pathlib module

⚠more in <https://docs.python.org/3/library/pathlib.html>



It prints the current directory where we are.

## Save objects into files

For that we use the pickle module.

We will then use 2 classes include in the module : Pickler and Unpickler

Syntax to create an object:



Syntax to return an object:



⚠ If there are several objects to save or retrieve, we rewrite several times the line checked by a ✶ red star.

# isinstance() function

A bit like the type function. Returns True if the specified object is of the specified type, otherwise False.

If the type parameter is a tuple, this function will return True if the object is one of type in the tuple.



# sorted() function

It returns a sorted list of the specified iterable object.

Syntax: sorted(iterable, key=key, reverse=reverse)

iterable= Required

key= Optional. A function to execute to decide the order.

reverse= Optional. A Boolean.



# map() function

The map() function executes a specified function for each item in a iterable.

The item is sent to the function as a parameter.

Syntax: map(function, iterables)

function= Required. The function to execute for each time.

Iterable= Required. A sequence, collection or an iterator object. We can send as many iterables as we like, just make sure the function has one parameter for each iterable.



⚠We convert the map into a list for readability. Without = <map object at 0x01F77028>

# Standard flows

3 standard flows=

* Standard entrance (ex: input())
* Standard output (used to display messages)
* The standard error (especially used when Python displays the *traceback* of an exception)

## Access standard flows

It is accessed through the sys module which offers several functions and variables to interact with the system.



To read and write in the standard streams, we use the read and write methods.

## Modify standard flows



To restore the old standard output, we must type: 

# Signals

Signals can be intercepted in our program. This allows us in particular to trigger a certain action if the program must close (ex: save objects in file, …)

Not all signals are on the same operating program. Consequently, we will give the example of only one signal: SIGINT, sent when the program is stopped.

## Intercept a signal



Press CTRL + C to stop the program

# random module

## Methods

### randrange()

returns a randomly selected element from the specified range.

Syntax: random.randrange(start, stop, step)

start= Optional. An integer specifying at which position to start. Default 0.

stop= Required. An integer specifying at which position to end.

step= Optional. An integer specifying the incrementation. Default 1.

### randint()

returns an integer number selected from the specified range

Syntax: random.randint(start, stop)

start= Required. An integer specifying at which position to start.

Stop= Required. An integer specifying at which position to stop.



### choices()

returns a list with the randomly selected element from the specified sequence.

Syntax: random.choices( sequence, weights=None, cum\_weights=None, K=1)

weights= Optional. A list where you can weigh the possibility for each value. Default None.

cum\_weights= Optional. Same as weights, only this time the possibility is accumulated.

K= Optional. An integer defining the length to the returned list.

### sample()

returns a list with a randomly selection of a specified number of items from a sequence.

Syntax: random.sample(sequence, K)

### shuffle()

takes a sequence and reorganize the order of the items.

Syntax: random.shuffle(sequence, function)

function= Optional.

### uniform()

returns a random floating number between 2 specified numbers.

Syntax: random.uniform(a, b)

### triangular()

returns a random floating number between 2 specified number, but we can weigh the possible outcome closer to one of the other 2 parameter values.

Syntax: random.triangular(low,high,mode)

# Password management

## Get a password

To get a password we use the getpass module.



The password that we type is invisible, however we do write.

## Encrypt a password

To encrypt a password, we can import the hashlib module.

This module offers us two list:

* algorithms\_guaranteed= algorithms guaranteed by python, the same from one platform to another (preferable to have portable programs)
* algrithms\_available

here we’ll be interested in sha1 (takes a bytes string as parameter). To get a bytes string from a raw string we can use the encode() method or put a lowercase b before opening the string. 

To obtain the encryption associated with this object, we have 2 possibilities:

* digest method=returns a bytes type containing our encrypted password.
* hexdigest method= returns a raw string containing a sequence of hexadecimal symbol (0-9 and A-F) / this one is better for file storage



## Generate random password

To generate a random password, we can use the secrets module.

Here we will see 2 of its function:

* choice: returns a random element from a sequence.
* token\_hex: generate a random string of hexadecimal symbol.



# The network

To communicate 2 applications, which can be on the same machine but also on remote machines. In this case, they connect through the local network or the internet.

## TCP protocol (transmission Control Protocol)

Used to connect 2 applications and make them exchange information.

This protocol is called “connection oriented”, that is to say that the applications are connected to communicate and that we can be sure, when we send information through the network, that is has been received by the other application. If the connection is broken for some reason, the application should reestablish the connection.

Unlike UDP (User Datagram Protocol) which sends information through the network without worrying about whether its target receives it correctly.

But UDP is faster than TCP.

## Clients and server

⚠Clients and servers must not be on the same application (windows)!!!!!!!!!!!!

There is usually one server and several clients

### The different steps (simplified)

The server:

1. wait for a connection from the client
2. accept a connection
3. exchange of information with the client
4. close connection

The client:

1. connect to the server
2. exchange information with the server
3. close connection

## establish a connection

For the client to connect to the server, we need 2 pieces of information:

* The host name, which identifies a machine on the internet or on a local network. Host names represent IP addresses more clearly.
* A port number, often specific to the type of information we are going to exchange. If we request a web connection, the browser will generally query the port 80 (http) or port 443 if it’s a secure connection (https). The port number is between 0 and 65535 and the numbers between 0 and 1023 are reserved by the system.

## Sockets

Sockets are objects that open a connection with a local or remote machine and exchange with it.

These objects are defined in the socket module.

### Build our socket

We use the socket construction. In the case of a TCP connection, it takes the following 2 parameters in order:

1. socket.AF\_INET: the address family, here they are Internet addresses.
2. socket.SOCK\_STREAM: the type of socket, SOCK\_STREAM for the TCP protocol.



### Connect the socket

In order for the server to wait for connections from the clients, we use the bind method. It takes one parameter: the tuple (hostname, port).

In our case the name of the host will be empty and the port will be the one we want between 1024 and 65535.



### Listen to our socket

We can specify the maximum number of connections that the socket can receive on its port without accepting them with the listen method.



### Accept a connection from the client

The accept method blocks the program as long as no client is connected.

This method returns two pieces of information:

* The connected socket that has just been created, the one that will allow us to dialogue with our client.
* A tuple representing the client’s IP address and connection port.

⚠There are therefore 2 ports in or history but the one used by the client to open his connection doesn’t particularly interest us.



### Creation of a client

We start by building our socket in the same way as that of the server.



### Connect the client

We use the connect method. It takes as parameter a tuple, like bind, containing the hostname and the number of the port identifying the server.

As our 2 applications are on the same machine, the hostname is localhost and we already know the port number on which we want to connect.



Our server and client are connected.

On the server side:



The first information is the client’s IP address (localhost redirects to this IP). The second one is the client’s output port, which does not interest us here.

### Communicate our sockets

We use the send method to send and recv to receive.

Server side: 

Client side:

The recv method takes the number of characters to read as a parameter. Usually, we pass it the value 1024.



### Close connection

We use the close method.



### The server, example code



### The client, example code



### A more elaborate server

Our server in the previous example has several problems:

* He can only accept one client
* We must respond immediately after the message form the customers, but this is not always possible in some cases.
* We can’t send and receive multiple messages in an unknown order

## The select module

The select module will allow us to question several customers while waiting for a message to be received, without paralyzing our program.

select will listen on a list of clients and return after a specified time. select returns the list of clients who have a message to receive. Just browse the customers and read the pending messages (thanks to rcv).

The function we are interested in has the same name as the associated module, select.

It takes 3 or 4 arguments:

* rlist= the list of sockets waiting to be read
* wlist= the list of sockets waiting to be written
* xlist= the list of sockets waiting for an error
* timeout=the amount of time the function waits before returning. If the parameter is not specified, the function returns as soon as a socket changes state.

We are trying to write sockets to a list and have select watch them, returning as soon as one of them is ready to be read. So, our program can receive messages in an unknown order from several clients.

### Server with select (better)





# Unit test with unittest

To test the correct functioning of a program.

## Basic structure of a unittest (test case)

🡨the easiest form

We import the unittest module. Then we define a class inheriting from unittest.TestCase. We can then define a test in a method whose name start with test.



## random.choice function test



1. We create a list of 0 to 10
2. We call the random.choice function on our list and get the return
3. We check that the element returned by random.choice is in our list. For that we raise an assertion.
4. Another way to do point 3

To run the test, we must execute the following instruction:



We should get something like this:



1. **.** = valid

**F** = bad result

**E** = error

1. Number of test performed in time in seconds
2. **OK** = all is good

or

**Number of success or failure**

## Test that an error is raised



## Initialization of tests

We can create a setup method which is called before each test method (so as not to have to repeat the same thing each time in the methods) ⚠ see Full recap of test code

There is also a teardown method which is called after each test.

## Full recap of test code (with a little bit more)



## mock.patch()

Common uses for Mock objects include:

* Patching methods
* Recording method calls on objects

Here, we’ll focus on patching.

We import mock or we can use unittest.mock



The patch decorators are used for patching objects only within the scope of the function they decorate. They automatically handle the unpatching for us, even if exceptions are raised. All of these functions can also be used in with statements or as class decorator.

Here we will see how to test the input, with the patch function and how to unit test inputs inside a function using the @patch decorator.

The patch function temporarily replaces the target object with different object during the test. The @patch decorator accepts a big amount of arguments, but here we will focus on side\_effect and return\_value. As we will talk only about mocking inputs, our target is the built-in function input, and the target for the patch decorator is “builtins.input”.

The side\_effect argument can accept a function to be called when the mock is called, an iterable or an Exception. Passing in an iterable is very useful to mock multiple inputs inside the testing function, because it will yield the next value every time it’s called.



The return\_value configure the value returned when the mock is called. It will always return the same value when the mock is called.



### Simple examples



Now we’ll change the function to have multiple inputs:





## Assert methods

|  |  |
| --- | --- |
| Method | Explanation |
| assertEqual(a, b) | a==b |
| assertNotEqual(a, b) | a!=b |
| assertTrue(x) | x is True |
| assertFalse(x) | x is False |
| assertIs(a, b) | a is b |
| assertIsNot(a, b) | a is not b |
| assertIsNone(x) | x is None |
| assertIsNotNone(x) | x is not None |
| assertIn(a, b) | a in b |
| assertNotIn(a, b) | a not in b |
| assertIsInstance(a, b) | isinstance(a, b) |
| assertNotIsInstance(a, b) | not isinstance(a, b) |
| assertRaises(exception, function, \*args, \*\*kwargs) | Check that the function raise the expected exception |
| assertRaisesRegex(exception, regexp, callable, \*args, \*\*kwargs) | Like assertRaises() but also tests that regex matches on the string representation of the raised exception |
| assertAlmostEqual(a, b, 7) | round(a-b, 7)==0 |
| assertNotAlmostEqual(a, b, 7) | round(a-b, 7)!=0 |
| assertGreater(a, b) | a>b |
| assertGreaterEqual(a, b) | a>=b |
| assertLess(a, b) | a<b |
| assertLessEqual(a, b) | a<=b |
| assertRegexpMatches(text, regex) | Test that a Regex search matches text |
| assertNotRegexpMatches(text, regex) | Test that a Regex search does not matches text |
| assertItemsEqual(a, b) | sorted(a)==sorted(b) |
| assertDictContainsSubset(a, b) | All the key/value in a exist in b |
| assertMultilineEqual(a, b) | Test that the multiline string a==b |
| assertSequenceEqual(seqA, seqB) | seqA== seqB |
| assertListEqual(listA, listB) | listA==listB |
| assertTupleEqual(tupA, tupB) | tupA==tupB |
| assertSetEqual(setA, setB) | setA==setB |
| assertDictEqual(dictA, dictB) | dictA==dictB |

## Automatic test discovery

Running test with unittest.main() can be pratical, but generally we’ll use the automatic test discovery.

This feature find all the unit tests contained in a package and run them.

### Launching unit tests from a directory

1. We open the window console (if we are in window)
2. We move to the directory of our tests(⚠ the test files must start with test!)

cd C:\User\...\...

1. We execute python with the option python -m unittest

If all goes well, we should see the tests run

We can also specify a test path to run:

1. test\_file: module’s name (name of the file without extension)
2. RandomTest: classe’s name of our module
3. test\_shuffle: Method’s name to execute

python -m unittest text\_file.RandomTest.test\_shuffle

⚠Unit tests must be independent. They should not depend on other tests to run.

# Debug with pdb

## Launch pdb

The native breakpoint() function starts the pdb debugger.

## Debugger Commands

Entering a blank line (enter) repeats the last command entered.

Exception: if the last command was a list command, the next 11 lines are listed.

Commands that the debugger doesn’t recognize are assumed to be Python statements and are executed in the context of the program being debugged. Python statements can also be prefixed with an !.

This is a powerful way to inspect the program being debugged; it is even possible to change a variable or call a function.

When an exception occurs in such a statement, the exception name is printed but the debugger’s state is not changed.

The debugger support aliases and they can have parameters, which allows one a certain level of adaptability to the context under examination.

Multiple commands may be entered on a single line, separated by ;;

If a file .pdbrc exists in the user’s home directory or in the current directory, it is read in and executed as if it had been typed at the debugger prompt. This is particularly useful for aliases. If both files exist, the one in the home directory is read first and aliases defined there can be overridden by the local file. .pdbrc can contain commands that continue debugging.

Optional arguments are enclosed in [ ]

help or h [command] = print the list of available commands/ help pdb= display the full documentation/ help exec= help on the ! command

where or w = Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame.

down or d [count] = Move the current frame count (default one) levels down in the stack trace (to a newer frame). d=d 1

up or u [count] = Move the current frame count (default one) levels up in the stack trace (to an older frame).

break or b [([filename:] lineno¦ function) [,condition]] =

With a lineno argument, set a break there in the current file.

With a function argument, set a break at the first executable statement within that function.

The line number may be prefixed with a filename and a :

Note that each breakpoint is assigned a number to which all the other breakpoint commands refer.

If a second argument is present, it is an expression which must evaluate to True before the breakpoint is honored.

Without argument, list all breaks, including for each breakpoint, the number of times that breakpoint has been hit.

tbreak [([filename:] lineno¦ function) [,condition]]= Temporary breakpoint, which is removed automatically when it is first hit. The arguments are the same as for [break](https://docs.python.org/3/library/pdb.html#pdbcommand-break).

clear or cl [filename: lineno ¦ bpnumber…] = With a filename:lineno argument, clear all the breakpoints at this line. With a space separated list of breakpoint numbers, clear those breakpoints. Without argument, clear all breaks.

disable [pbnumber…]= Disable the breakpoints given as a space separated list of breakpoint numbers.

enable [pbnumber…]= Enable the breakpoints specified.

ignore bpnumber [count]= Set the ignore count for the given breakpoint number. If count is omitted, the ignore count is set to 0. A breakpoint becomes active when the ignore count is zero. When non-zero, the count is decremented each time the breakpoint is reached and the breakpoint is not disabled and any associated condition evaluates to true.

condition bpnumber [condition]= Set a new condition for the breakpoint, an expression which must evaluate to true before the breakpoint is honored. If condition is absent, any existing condition is removed; i.e., the breakpoint is made unconditional.

commands [bpnumber]= Specify a list of commands for breakpoint number bpnumber. Type a line with just end to terminate the command. To remove all commands from a breakpoint, type commands end.

step or s = Execute the current line, stop at the first possible occasion.

next or n= Continue execution until the next line in the current function is reached or it returns difference step/next = step: stop inside a call function next: only stop at the next line in the current function

until or unt [lineno]= Without argument, continue execution until the line with a number greater than the current one is reached. With a line number, continue execution until a line with a number greater or equal to that is reached. In both cases, also stop when the current frame returns.

return or r= Continue execution, only stop when a breakpoint is encountered.

continue or cont or c = Continue execution, only stop when a breakpoint is encountered.

jump or j lineno= Set the next line that will be executed. Only available in the bottom- most frame. This lets you jump back and execute code again, or jump forward to skip code that you don’t want to run. (Not all jumps are allowed: ex= impossible to jump in the middle of a for loop)

list or l [first [,last]]= List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing. With . as argument, list 11 lines around the current line. With one argument, list 11 lines around at that line. With two arguments, list the given range; if the second argument is less than the first, it is interpreted as a count.

The current line in the current frame is indicated by ->. If an exception is being debugged, the line where the exception was originally raised or propagated is indicated by >>, if it differs from the current line.

longlist or ll = list all source code for the current function or frame. Interesting lines are marked as for list.

args or a= Print the argument list of the current function.

p expression= Evaluate the expression in the current context and print its value.

pp expression = Like the [p](https://docs.python.org/3/library/pdb.html#pdbcommand-p) command, except the value of the expression is pretty- printed using the [pprint](https://docs.python.org/3/library/pprint.html" \l "module-pprint" \o "pprint: Data pretty printer.) module.

whatis expression = Print the type of the expression.

source expression= Try to get source code for the given object and display it.

display [expression]= Display the value of the expression if it changed, each time execution stops in the current frame. Without expression, list all display expressions for the current frame.

undisplay [expression] = Do not display the expression any more in the current frame. Without expression, clear all display expressions for the current frame.

interact = Start an interactive interpreter (using the [code](https://docs.python.org/3/library/code.html#module-code) module) whose global namespace contains all the (global and local) names found in the current scope.

alias [name[command]]= Create an alias called name that executes command. The command must not be enclosed in quotes. Replaceable parameters can be indicated by %1, %2, and so on, while %\* is replaced by all the parameters. If no command is given, the current alias for name is shown. If no arguments are given, all aliases are listed.

Aliases may be nested and can contain anything that can be legally typed at the pdb prompt. Note that internal pdb commands can be overridden by aliases. Such a command is then hidden until the alias is removed. Aliasing is recursively applied to the first word of the command line; all other words in the line are left alone.

unalias name= delete the specified alias.

run [args…]

restart [args…]= Restart the debugged Python program. If an argument is supplied, it is split with [shlex](https://docs.python.org/3/library/shlex.html" \l "module-shlex" \o "shlex: Simple lexical analysis for Unix shell-like languages.) and the result is used as the new [sys.argv](https://docs.python.org/3/library/sys.html" \l "sys.argv" \o "sys.argv). History, breakpoints, actions and debugger options are preserved. [restart](https://docs.python.org/3/library/pdb.html#pdbcommand-restart) is an alias for [run](https://docs.python.org/3/library/pdb.html#pdbcommand-run).

quit or q= Quit from the debugger. The program being executed is aborted.

debug code= Enter a recursive debugger that steps through the code argument (which is an arbitrary expression or statement to be executed in the current environment).

retval = Print the return value for the last return of a function.

Footnotes = Whether a frame is considered to originate in a certain module is determined by the \_\_name\_\_ in the frame global.

# Threading

Allows us to do parallel programming, that is, several code instructions wil be executed almost at the same time.

This module defines the following functions:

## Functions

active\_count( )= returned the number of Thread objects currently alive. Returned count is equal to the length of the list returned by enumerate().

current\_thread( )= return the current Thread object, corresponding to the caller’s thread of control ( if not created through the threading module, a dummy thread object with limited functionality is returned.

excepthook(args,/)=Handle uncaught exceptions raised by Thread.run().

The args argument has the following attributes:

* exc\_type: Exception Type
* exc\_value: Exception value, can be None
* exc\_traceback: Exception traceback, can be None
* thread: Thread which raised the exception, can be None

get\_ident( )= return the “thread identifier” of the current thread (nonzero integer). It’s value has no direct meaning 🡪 used to index a dictionary of thread-specific data.

enumerate( )= return a list of all Thread objects currently alive.

main\_thread( )= return the main Thread object

settrace(func)= Set a trace function for all thread from the threading module. The func will be passed to sys.setprofile() for each thread, before its run() method is called.

stack\_size([size])= return the thread stack size used for creating new threads. The optional size argument specifies the stack size to be used for subsequently created thread, and must be 0 or a positive integer value of at least 32,768 (32KiB).

This module also defines the following constant:

TIMEOUT\_MAX= the maximum value allowed for the timeout parameter of blocking function(Lock.acquire(), RLock.acquire(), Condition.wait(), etc…). Specifying a timeout greater than this value will raise an OverflowError.

## Thread Objects

The Thread class represents an activity that is run in a separate thread of control. We can only override the \_\_init\_\_ and run() method of this class.

Once a thread object is created, its activity must be started by calling the start() method (invokes the run() method in a separate thread of control).

Once the thread’s activity is started, the thread is considered “alive” (when its run() method terminate, it stops being alive). The is\_alive() method tests whether the thread is alive.

If the subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.\_\_init\_\_()) before doing anything else to the thread.

join(timeout=None)= wait until the thread terminates

get/setName()= old getter/setter API for name; use it directly as a property instead.

## Lock objects

RLock and Lock are almost the same thing except that Lock can only be acquired once, until is released. On the other hand, RLock can be acquired multiple times(recursive).

acquire()= lock

release()= unlock

Locks support the context management protocol:

The acquire() method will be called when the block is entered, and release() when the block is existed.



## Small example



These are other objects to discover with this module but we will stop here, for more information got to the site.

# \_\_main\_\_ - Top level script environment

“\_\_main\_\_” is the name of the scope in which top-level code executes. A module’s \_\_name\_\_ is set equal to “\_\_main\_\_” when read from standard input, a script, or from an interactive prompt.

A module can discover whether or not it’s running in the main scope by checking it’s own \_\_name\_\_, which allows a common idiom for conditionally executing code in a module when it’s run as a script or with python -m but not when it’s imported:



For a package, the same effect can be achieved by including a \_\_main\_\_.py module, the contents of which will be executed when the module is run with -m.

# Asynchronous programming with asyncio

Asyncio allows us to split our program into tasks, which are then executed in parallel.

A task is a function.



* we add the word async in front of the task and main functions
* await asyncio.sleep(1) indicates that the task should pause for a second and also signals Python that it can run another task while waiting.
* await asyncio.gather(task(1), task(2), task(3)) create 3 asynchronous tasks.
* asyncio.run(main()) run main()

an asynchronous task is called in Python a coroutine.

## Example of use





* async in front of with = indicates that the context is asynchronous

# GUI with Tkinter

Built-in module in Python

## Minimum code



* we create an object of class Tk. Most of the time this object will be the main window of our interface.
* We create a label, i.e., a graphic object displaying text
* The pack() method is used to display the object
* We call the mainloop() method of our root window. It only returns when we close the window.

## widgets

Our graphic objects (buttons, text fields, checkbox,…) are called widgets.

We can specify several options when creating our widgets. There are several common to most widgets (their color fg, that of the background bg,…).

The options can be modified during the creation of the widget but also after.



Just pass the name of the option between brackets.

⚠For a widget to appear, it must take the window as first parameter and call the pack() method.

### labels

Display text (can’t be modified by the user).



### buttons

Widget that you click to trigger actions or command



### An entry line

Text area in which the user can write.



In the first line, we create a tkinter variable. It contains the text of our Entry (we don’t have to put it but allows us to save the text or link an action).

There is also the Text widget which represents a multi-line text field.

### checkboxes

Checkboxes are defined in the Checkbutton class. Again, we use a variable (IntVar) to monitor the selection of the box.



We can then control the state of the checkbox by querying the variable: 

If the box is checked, the value returned by the variable will be 1, otherwise 0.

### radio buttons

Set of checkboxes (only one checkbox can be checked at the time).

To create a group of buttons, they must be linked to the same Tkinter variable.

When the user changes the selected button, the value of the variable also changes according to the value option associated with the button.



To retrieve the value associated with the currently selected button, “query” the variable:



### The drop-down list

List in which one or more element are selected.



The insert method takes 2 parameters:

1. Position of the element
2. Str element



To insert elements at the end of a list, we use the END constant of tkinter.

To access the selection, we use the curselection() method of the list. It returns a tuple of character string, each being the position of the selected element.



## Organize our widgets in the window

There are several widgets which serve to contain other. One of them is called Frame.

For a widget to appear in a Frame, we must specify the Frame as its parent when creating.



By specifying side=”top”, we ask that the element be place at the top of its parent.

fill allows the widget to fill it’s parent, in width(X), height(Y) or both(BOTH).

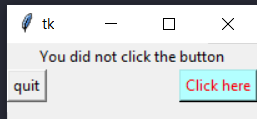
There is also the LabelFrame widget, a frame with a title, it’s built like a Frame but take the text option which will be its title.

For more widget go to the official site.

## commands

From the moment we leave the framework of a test, it’s preferable to write the code in a class.

We can make it inherit from Frame, which means that our class will be a widget too.



* We create a class that will contain the whole window and inherits Frame.
* In the window constructor, we call the frame and display (pack) the frame.
* Still in the constructor, we create the different widgets of the window. They are positioned and displayed as well.
* We create a click() method, which is called when we press button\_click. It doesn’t take any parameter and updates the text contained in the self\_message label.
* We create the Tk window which is the parent object of the interface that we then instantiate.
* We enter the mainloop loop.
* We destroy the window

# Pause a program



# Virtual environments

See Pip or Using requirements file

Folder containing a copy of our installed Python version.

## Create a virtual environment

* Open a command line and go to the folder where we want to create our virtual environment.
* Enter : python -m virtualenv environment\_name

## Activate our virtual environment

Under Window, in the same folder, enter the command: environment\_name\scripts\activate

To deactivate the virtual environment, enter: deactivate

# Using requirements file

See Pip or Virtual environments

Instead of installing packages individually, pip allows us to declare all dependencies in a requirements file.

Example= requests==3.5.1

In a requirements.txt file

six==1.15.0

This command will export pip freeze as a requirements.txt:

pip freeze > requirements.txt

Once we’ve got our requirements file, we can head over to a different computer or virtual environment and run the following:

pip install -r requirements.txt

That’s assuming we are working in the directory containing requirement.txt. This tells pip to install the specific versions of all the dependencies listed.

To upgrade our installed package, we run the following:

pip install –upgrade -r requirements.txt

# Distribute our Python program with cx\_Freeze

There are several way to produce executable files that we can distribute that include everything we need.

In Windows, we have to enclose our files with the extension .py in a .exe accompanied by .dll files. Cx\_freeze is one of the tools to achieve this goal (Pyinstaller, Py2exe,… too).

Download cx\_freeze preferably in a virtual environment.

## Use the cx\_freeze

⚠don’t forget to run in the correct folder!

In the command line enter: cx\_freeze file\_name.py

The executable will be in the dist folder.

⚠To share it, make sure to copy everything in the dist folder along with our program. Otherwise, our executable might not launch properly.

## The setup.py file

The setup.py file is responsible for creating the executable of our program. Also allows to transform several Pythons file at the same time.

A basic setup.py file contains this code:



**\***takes in parameter the path of the files

On window, in the command line = setup.py build

# keyboard module

used to control the keyboard

## functions

### write()

It writes the content to output



### send() alias= press\_and\_release()

Sends OS events



### press() and release()

press or release a button



### is\_pressed()

Return True if the key is pressed

### call\_later()

Calls a function and wait some time.



### wait()

Block the program execution until the given hotkey is pressed or, if given no parameters, blocks forever.



### start\_recording() and stop\_recording()

Starts recording all keyboard events into a global variable, or the given queue if any.

Stops the global recording of events and returns a lists of the events captured.



### record()

records all keyboard events until the user presses the given hotkey.



### play()

Play a sequence of recorded events, maintaining the relative time intervals. If speed\_factor is <=0 the actions are replayed as fast as the OS allows.

Pairs well with “record()”.



### add\_abbreviation()

Register a hotkey that replaces one typed text with another.



### add\_hotkey() and remove\_hotkey()

Invokes a callback every time a hotkey is pressed.

Remove a previously hooked hotkey.



# put a default interpreter on a python script

To put a default interpreter (ex= virtual environment) on a python script, we place at the top of the file:

#! path/to/env/scripts/python

# Specify the working encoding

To specify the working encoding, we place at the top of the file:

# -\*- coding: encoding -\*-



# requests module

The requests module allows us to send HTTP requests using Python.

The HTTP request returns a response object with all the response data (content, encoding, status, … ).

Syntax : requests.methodname(params)

## Methods

delete(url, args) = Sends a delete request to the specified url.

get(url, params, args) = Sends a get request to the specified url.

head(url, args) = Sends a Head request to the specified url.

patch(url, data, args) = Sends a patch request to the specified url.

post(url, data, json, args) = Sends a post request to the specified url.

put(url, data, args) = Sends a put request to the specified url.

request(method, url, args) = Sends a request of the specified method to the specified url

⚠ Everywhere the url is required !!!

The parameters can be url(required), or (optional args) :

* allow\_redirects = A Boolean to enable/disable redirection.
* Auth = A tuple to enable a certain HTTP authentication.
* cert = A string or tuple specifying a cert file or key.
* cookies = A dictionary of cookies to send to the specified url.
* headers = A dictionary of HTTP headers to send to the specified url.
* proxies = A dictionary of the protocol to the proxy url.
* stream = A Boolean indication if the response should be immediately download (False) or streamed (True).
* timeout = A number or a tuple, indicating how many seconds to wait for the client to make a connection and/or send a response.
* verify = A Boolean or a string indication to verify the server TLS certificate or not.

## requests response object

The response object contains the server’s response to the HTTP request.



### properties and methods

apparent\_encoding = returns the apparent encoding.

close() = closes the connection to the server

content = returns the content of the response, in bytes.

cookies = returns a CookieJar object with the cookies sent back from the server.

elapsed = returns a time data object with the time elapsed from sending the request to the arrival of the response.

encoding = returns the encoding used to decode r.text .

headers = return a dictionary of response headers.

history = returns a list of response object holding the history request (url).

is\_permanent\_redirect = returns True if the response is permanent redirected url otherwise False.

is\_redirect = return True if the response was redirected, otherwise False.

iter\_content() = iterates over the response.

iter\_lines() = iterates over the lines of the response.

json() = returns a JSON object of the result (if result not written in JSON format raises an error)

links = returns the header links.

next = returns a PreparedRequest object for the next request in a redirection.

ok = returns True if status\_code is less than 400, otherwise False.

raises\_for\_status = if an error occur, this method returns a HTTPError object.

reason = returns a text corresponding to the status code.

request = returns the request object that requested this response.

status\_code = returns a number that indicates the status (200 is ok, 404 is Not Found).

text = returns the content of the response, in unicode.

url = returns the URL of the response.

# Creating Web APIs with Flask

## minimal Flask App



## variable rules

We can add variable sections to a URL by using <variable\_name>.

The function receives the variable as a keyword argument.



When we run the above code and open the browser and go to <http://localhost:5000/Maria/> we’ll see the output as Hello Maria and when we go to <http://localhost:5000/9/> the output will be Incremented number is 10.

## Redirection behavior



This result in 2 different behaviors:

* For the home endpoint, if we access the URL without the trailing slash, then Flask redirects us to the URL with the trailing slash.
* For the contact endpoint, if we access the URL with the trailing slash, then it will result in status 404 not Found.

## returns status code

We can return the status code along with the Response by specifying the status code as follows.



The response to this URL will be Would you like some tea? with 418 as the status code instead of the usual 200.

## Before request

We can specify a function which should always execute before the request is processed by using @app.before\_request decorator.



## app.run() parameter

* host and port, we saw them : minimal Flask App
* debug = if set to True, the server will automatically reload on code changes and show an interactive debugger in case of unhandled exceptions. Default False.
* use\_reloader = if set to True, the server will automatically restart when code changes. Default is False.
* threaded = if set to True, the process will handle each request in a separate thread. Default is False.
* ssl\_context = SSL Context for the connection. Expects ssl.SSLContext, a tuple in the form (Cert\_file, pkey\_file), or a string ‘adhoc’ if the server should automatically create the context. Default is None i.e., SSL is disabled.

This is used when we want to host the Flask application on HTTPS instead of HTTP.

## Blueprints

Blueprints allows us to separate various endpoint into subdomains.

**home.py**



**contact.py**



**app.py**



When we go to <http://localhost:5000/home/hello>, the output will be Hello from home page and when we visit <http://localhost:5000/contact/hello>, the output will be Hello from contact page.

## logging

We can use the following methods to log statements in a Flask Application:

* app.logger.debug(‘This is a DEBUG message’)
* app.logger.info(‘This is a INFO message’)
* app.logger.warning(‘This is a WARNING message’)
* app.logger.error(‘This is an ERROR message’)

## query string



? starts the query string

q is the 1st parameter

= separates/assigns a value to the parameter

query+string is the value assigned to the q parameter.

Space in query string are replaced with + or %20.

We use & to separate parameter and values.

https//…/query?foo=foo&bar=bar&baz&baz&title=query+string+with+Flask

### get query string



This example returns the value of the “user” parameter passed in the query string.

# NumPy

NumPy is the fundamental package for scientific computing in Python.

## The basics

NumPy’s main object is the homogenous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy dimensions are called axes.

For example, the coordinates of a point in 3D space [1,2,1] has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example below, the array has 2 axes. The first axis has length of 2, the second axis has a length of 3.



NumPy’s array class is called ndarray, also known by the alias array. numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional array and offer less functionality. The more important attributes of an ndarray object are :

* ndarray.ndim = The number of axes (dimension) of the array.
* ndarray.shape = the dimensions of the array. It’s a tuple of integer indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.
* ndarray.dtype = an object describing the type of the elements in the array. One can create or specify dtype’s using standart Python types. Additionaly NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.
* ndarray.itemsize = The size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It’s equivalent to ndarray.dtype.itemsize.
* ndarray.data = The buffer containing the actual elements of the array. Normally, we don’t need to use this attribute because we will access the element in an array using indexing facilities.



## Array creation

There are several ways to create an array.

⚠See documentataion of = array, zeros, zeros\_like, ones, ones\_like, empty, empty\_like, arange, linspace, numpy.random.Generator.rand, numpy.random.Generator.randn, fromfunction, fromfile

### With regular Python list/tuple using array function

The type of the resulting array is deduced from the type of the elements in the sequence.



⚠frequent error = 

array transforms sequence of sequence into two-dimensional array, sequence of sequence of sequence into three-dimensional array, and so on.



The type of the array can be specified at creation time:



### With zeros, ones, empty functions

The functions zeros creates an array full of zeros, the function ones creates an array full of ones, and the function empty creates an array whose initial content is random and depends on the state of the memory.

By default, the dtype of the created array is float64.





### with arange or linspace

To create sequences of number, NumPy provides the arange function which is analogous to the P ython built in range, but returns an array.



When arange is used with floating point arguments, it’s generally not possible to predict the number of elements obtained, due to the finite floating point precision. For this reason, it’s usually better to use the function linspace that receives as an argument the number of elements that we want, instead of the step:



## printing array

When we print an array, it displays with the following layout:

* The last axis is printed from left to right.
* The second to last is printed from top to bottom.
* The rest are also printed from top to bottom, with each slice separated from next by an empty line.

## Adding, removing, and sorting elements

We use np.sort() to sort elements. We can specify the axis, kind, and order when we call the function.



In addition to sort, which returns a sorted copy of an array, we can use:

* argsort, which is an indirect sort along a specified axis
* lexsort, which is an indirect stable sort on multiple keys
* searchsorted, which will find elements in a sorted array
* partition, which is a partial sort

We can concatenate array with np.concatenate().



In order to remove elements from an array, it’s simple to use indexing to select the elements that we want to keep.

## Basic operation

Arithmetic operators on array apply elementwise. A new array is created and filled with the result.



Unlike in many matrix languages, the product operator \* operates elementwise in NumPy arrays. The matrix product can be performed using the @ operator (in python >=3.5) or the dot function or method:



Some operations, such as += and \*=, act in place to modify an existing array rather than create a new one.

When operating with arrays of different types, the type of the resulting array corresponds to the more general or precise one (a behavior known as upcasting).

By default, these operations apply to the array as though it were a list of number, regardless of its shape. However, by specifying the axis parameter we can apply an operation along the specified axis of an array:



## Universal functions

NumPy provides familiar mathematical functions such as sin, cos, exp,…

These are called “universal functions” (unfunc). Within NumPy, these functions operate elementwise on an array, producing an array as output.



There are plenty of other functions.