THE BIOROBOTICS INSTITUTE

Scuola Superiore Sant'Anna

Introduction to the Python programming language

Ugo Albanese
Egidio Falotico

{e.falotico, u.albanese}@santannapisa.it

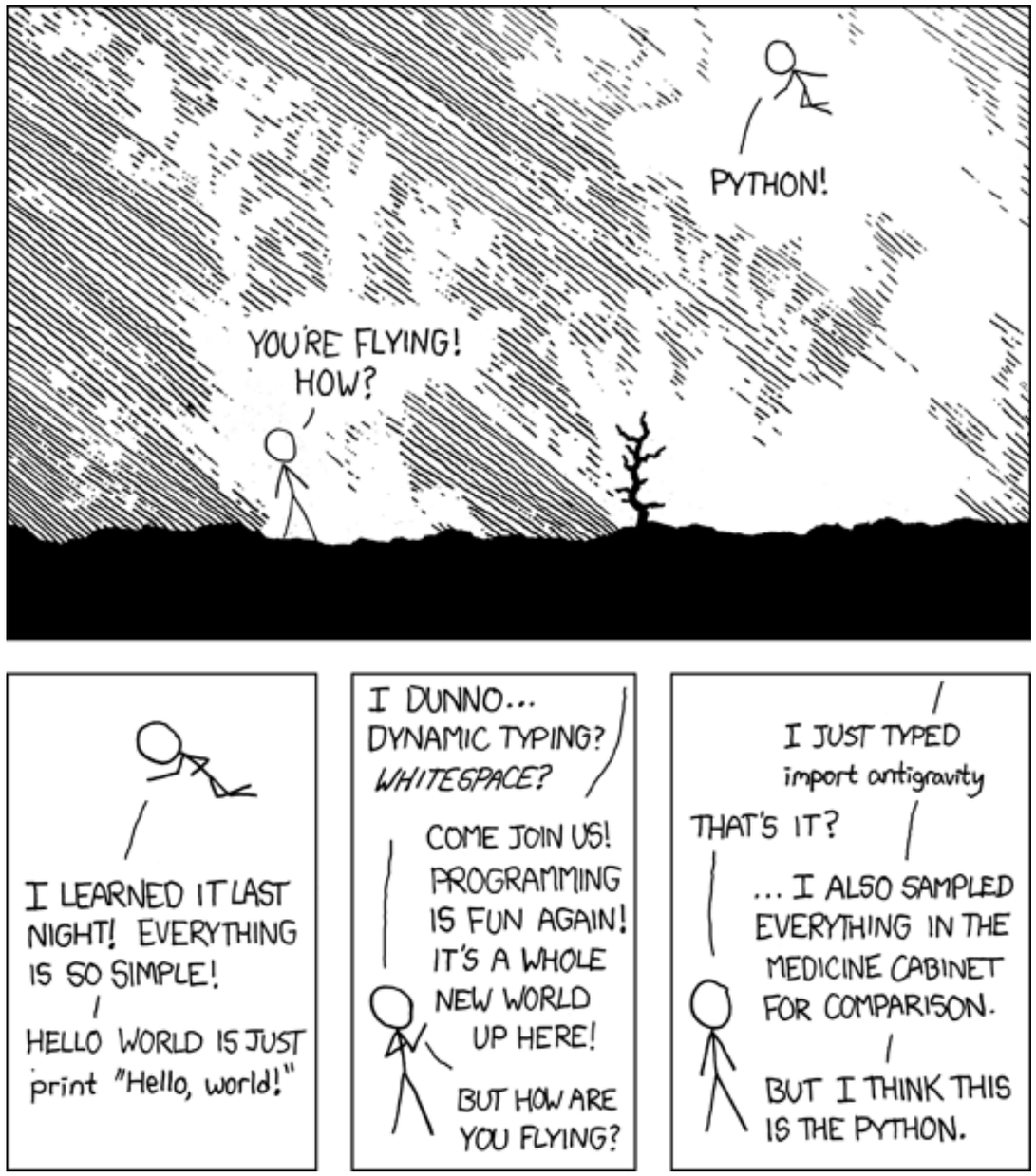
The Python programming Language

- A general-purpose programming language that blends procedural, functional, and object-oriented paradigms
- https://www.python.org/

- Created by Guido Van Rossum in 1991 (The *BDFL, Benevolent Dictator For Life*)

- Works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc)

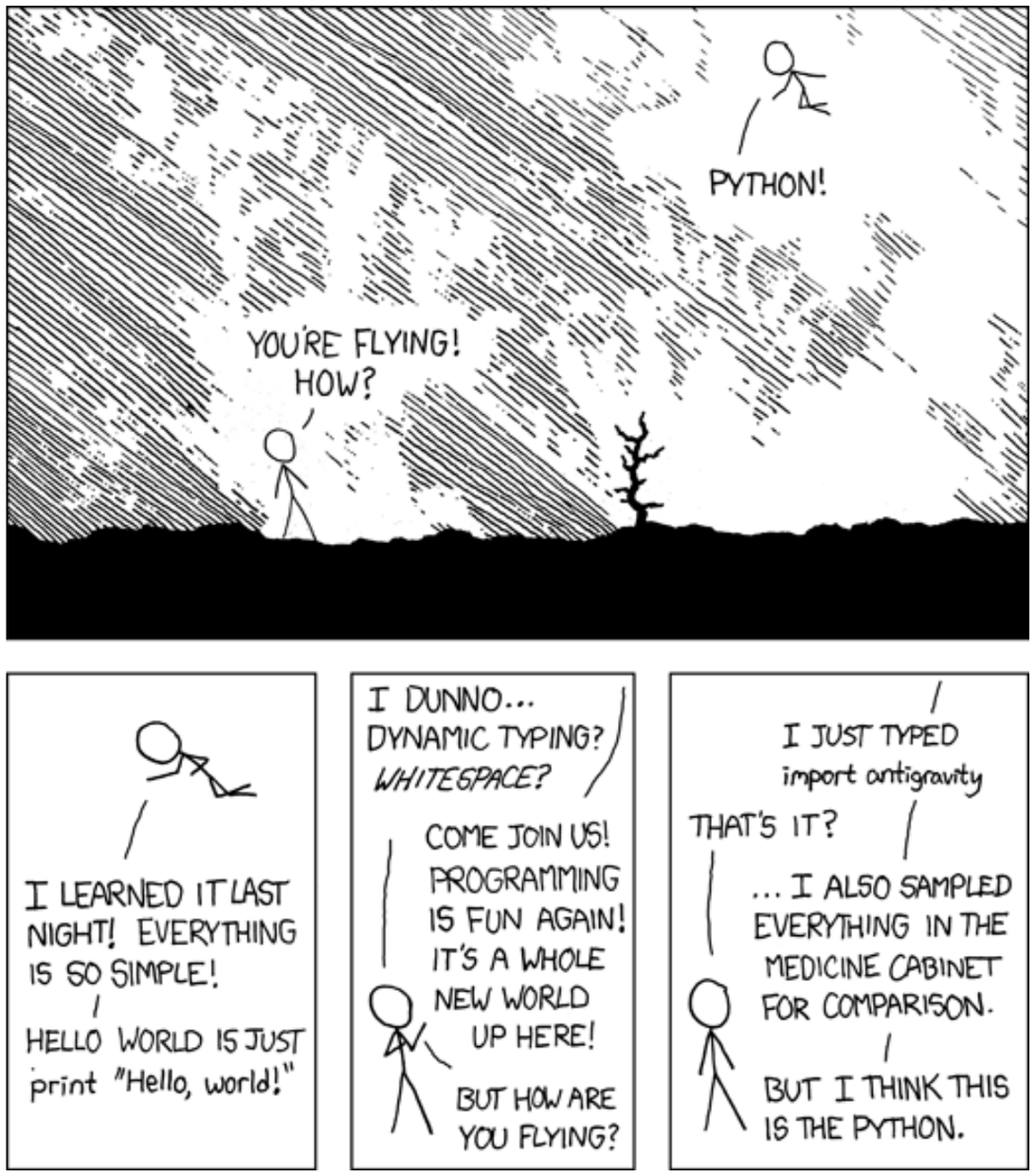
- Main uses:
  - Web development (server-side),
  - Scientific computing (scipy)
  - Data Science and ML
  - System scripting
  - IoT



https://xkcd.com/353/

The Python programming Language

- Simple syntax similar to the English language.
- Runs on an interpreter system, no need to compile, the code can be executed as soon as it is written. Easy to prototype new ideas.
- Relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. C-derived programming languages use curly-brackets instead.
- Latest version is Python 3.14



https://xkcd.com/353/

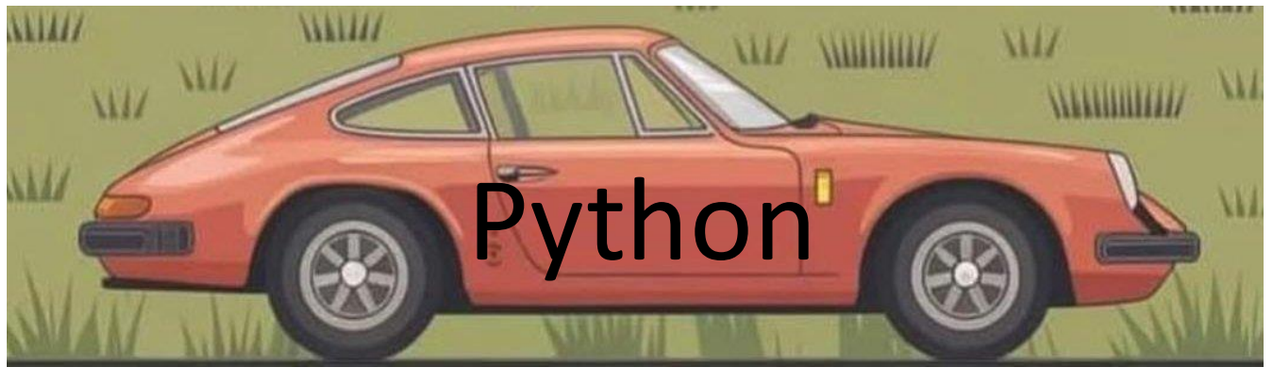
The Python programming Language

- Even though Python is an interpreted language, it can be fast.

- The most performant Python libraries are implemented in a lower level compiled programming language such as C/C++:
  - numpy (numerical computing)
  - scipy (scientific computing)
  - PyTorch (neural networks)

- Exposing the functions via a Python interface (bindings, API)

- Most common tools:
  - ctypes
  - CFFI
  - Boost.Python
  - pybind11



The Python programming Language: **Hello World**

Open a shell and run the python interpreter

```
brairlab@brairlab-vm:~$ python3
Python 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python Tutorial

https://docs.python.org/3/tutorial/index.html

The interpreter is ready to accept commands

```
>>> print("Hello World")
Hello World
>>>
```

Python scripts are usually written in .py files, then run by an interpreter.

```
brairlab@brairlab-vm:~$ echo "print('Hello World')" >> hello.py
brairlab@brairlab-vm:~$ python hello.py
Hello World
```

The Python programming Language: **Syntax**

- **Indentation** indicates a block of code.
- Indentation refers to the spaces at the beginning of a code line (4 per nesting level)
- In python the indentation is *very* important, it's not only for readability (like in other languages)

```
if 4 > 3:
    print("Four is greater than three") # CORRECT
else:
print("Impossible") # WRONG missing indentation
```

- Comments starts with a #, no multiline (multiline strings triple quotes as workaround)

```
# THIS IS A COMMENT
# THIS TOO

'''
Unassigned Multiline strings are ignored by the interpreter.

Use them as multiline comments
'''
```

The Python programming Language: **Data Model**

- **Objects** are Python's abstraction for data.
- *All data* in a Python program is represented by objects or by relations between objects.
- Every object has an **identity**, a **type** and a **value**.
- An object's **identity** *never changes* once it has been created; you may think of it as the object's address in memory (CPython).
- The `is` operator compares the identity of two objects; the `id()` function returns an integer representing its identity.

Python Data Model

https://docs.python.org/3/reference/datamodel.html

Alternative Python implementations

https://www.python.org/download/alternatives/

The Python programming Language: **Data Model - Type**

- An object's **type** determines the operations that the object supports (e.g., "does it have a length?") and defines the possible values for objects of that type.

- The `type()` function returns an object's type (which is an object itself).

- Like its *identity*, an object's type is also unchangeable.

Python Data Model

https://docs.python.org/3/reference/datamodel.html

The Python programming Language: **Data Model - Value**

- The **value** of some objects *can* change.
  - Objects whose value can change are said to be *mutable*
  - Objects whose value is unchangeable once they are created are called immutable.

Python Data Model

https://docs.python.org/3/reference/datamodel.html

- An object's mutability is determined by its type; (e.g. numbers, strings and tuples are immutable, dicts, lists are not)

- Objects are *never explicitly* destroyed
  - when they become un-reachable, they *may be garbage-collected*.

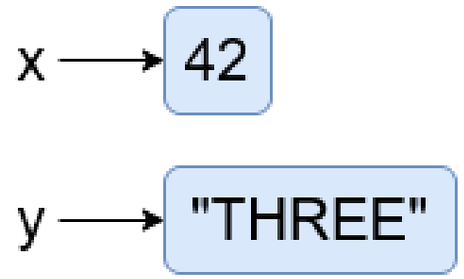
Garbage collection

- wiki
- Python glossary

The Python programming Language: **Variables**

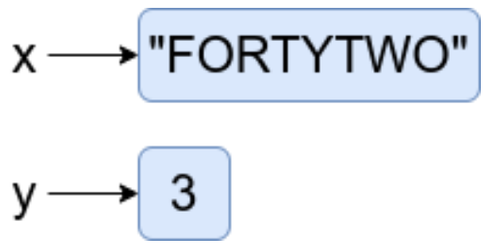
- **Variables** are containers for storing data values
- No need of declaration
- Created the moment you first assign a value to it

```
x = 42
y = "THREE"
print(x)
print(y)
```



- No type declaration, type can be changed after first assignment

```
x = "FORTYTWO"
y = 3
```



- Assign different/same value(s) to multiple variables

```
x, y, z = "a", "b", "c"

x = y = z = "a"
```

The Python programming Language: **Variables**

Rules for Python variable names:

- must start with a letter or the underscore character
- cannot start with a number
- can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- case-sensitive (foo, Foo and FOO are three different variables)

```
#Legal variable names:
myvar = "SSSA"
my_var = "SSSA" # snake case, pythonic

MyVar = "SSSA" # pascal case, classes
_my_var = "SSSA"

myVar = "SSSA" # camel case
MYVAR = "SSSA"
myvar2 = "SSSA"

#Illegal variable names:
2myvar = "SSSA"
my-var = "SSSA"
my var = "SSSA"
```

Coding style guide
https://docs.python.org/3/tutorial/controlflow.html#intermezzo-coding-style

PEP8:
https://www.python.org/dev/peps/pep-0008/

The Python programming Language: **Built-in Data Types**

The following data types are built-in, in these categories:

Text Type:	`str`
Numeric Types:	`int, float, complex`
Sequence Types:	`list, tuple, range`
Mapping Type:	`dict`
Set Types:	`set, frozenset`
Boolean Type:	`bool (True/False)`
Binary Types:	`bytes, bytearray, memoryview`

Use the function `type()` to get the data type of any object.

The Python programming Language: **Built-in Data Types**

Use the function `type()` to get the data type of any object.

```
x = 3
print(type(x)) # int
```

The data type is set when you assign a value to a variable:

```
x = 3.14
print(type(x)) # float, was int

x = "Hello World"
print(type(x)) # str, was float
```

Each data type has a "constructor" function:

- Create new instances of the type
- Convert type ("casting")

```
x = float(3)
print(x) # 3.0
print(str(x)) # str "3.0"
```

The Python programming Language: **Numbers**

Three numeric types in Python:

- `int` - plain integers (*unlimited* length)
- `float` - also in scientific notation e.g. `1.5e2`
- `complex` – complex numbers

reference
https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

```
x = 10   # int
y = 2.7  # float
z = 1+3j # complex
```

Convert from one type to another with:

- `int()`
- `float()`
- `complex()`

Random numbers with `random`
https://docs.python.org/3/library/random.html

The Python programming Language: **Numbers**

Operation	Result
`x + y`	sum of x and y
`x - y`	difference of x and y
`x * y`	product of x and y
`x / y`	quotient of x and y
`x // y`	(integer) quotient of x and y
`x % y`	remainder of x / y
`-x`	x negated
`+x`	x unchanged
`abs(x)`	absolute value or magnitude of x
`int(x)`	x converted to integer
`float(x)`	x converted to floating point
`complex(re,im)`	a complex number with real part re, imaginary part im. im defaults to zero.
`c.conjugate()`	conjugate of the complex number c. (Identity on real numbers)
`divmod(x, y)`	the pair (x // y, x % y)
`pow(x, y)`	x to the power y
`x ** y`	x to the power y

The Python programming Language: **Strings**

Textual data is handled with str objects, or strings, immutable sequences of Unicode code points.
Literals of type `str`, are surrounded by either single quotation marks, or double quotation marks.

reference
https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

```
"Hello World" # double quotation marks
'Hello World' # single
```

Being a Sequence type, strings can be indexed and sliced:

```
hello = "Hello World!"
hello[0] # "H"
hello[-1] # "!"
hello[1:3] # "el" (start:end-1:step)
```

The length is computed with the `len()` function

```
len(hello) # 12
```

The Python programming Language: **Strings**

Some built-in string method:

```
" between_spaces ".strip() # "between_spaces"

'CAPITAL'.lower() # "capital"

"small".upper() # "SMALL"

"Hello World".replace("World", "Earth") # "Hello Earth"

"comma,separated".split(",") # ["comma", "separated"]
```

reference
https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Membership test (`in, not in`):

```
hello_str = "Hello World!"

"Hello" in hello_str # True
"Earth" in hello_str # False
"Earth" not in hello_str # True
```

Concatenation

```
"Con" + "catenation" -> "Concatenation"
```

The Python programming Language: **Strings**

Formatting:

- old style - modulo operator % (printf-like)
- new style – `formatted string literals` (f-strings)

reference
https://docs.python.org/3/library/stdtypes.html#string-formatting-operations

```
# old style – format_str % values
# %d -> int, %s -> str, %f -> float

"Hello %s" % "World"
"The length is %d" % len(hello) # integer
```

A *formatted string literal* or *f-string* is a string literal that is prefixed with `'f'` or `'F'`. These strings may contain replacement fields, which are expressions delimited by curly braces {}.

Format specification mini-language

https://docs.python.org/3/library/string.html#formatspec

```
value = 12.34567
f"result: {value}"
'result: 12.34567'

f"result: {value:.3f}" # 3 digits after the decimal point
'result: 12.346'
```

The Python programming Language: **Booleans**

- A type with only two values:
  - `True`
  - `False`

reference
https://docs.python.org/3/library/stdtypes.html#truth-value-testing

- Any object can be tested for truth value, the following are considered false:
  - `False`
  - Empty values such as `()`, `[]`, `{}`, `""`
  - zero of any numeric type: `0, 0.0, 0j, Decimal(0), Fraction(0, 1)`
  - `None`

- Boolean Operators:
  - `and`
  - `or`
  - `not`

The Python programming Language: **Booleans**

Comparison operators (chainable):

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
`is`	object identity
`is not`	negated object identity

reference
https://docs.python.org/3/library/stdtypes.html#comparisons

The Python programming Language: **Lists**

A list is a (heterogeneous) collection of items:

- Ordered
- Mutable
- Allows duplicates

```
a_list = [1, 2, 3]
str_list = ["a", "b", "a"]
weird_list = ["1", 1, 1.0]

# access an element
a_list[2] # 3

# set a value
str_list[2] = "c" # ["a", "b", "c"]

# membership test
"1" in weird_list # True

# add item to list (end)
a_list.append(4) # [1, 2, 3, 4]

# remove item from list
weird_list.remove(1.0) # ["1", 1]

# find the index of the first specified element
a_list.index(2) # 1 -> a_list[1] is 2
```

Operations supported: sequences
https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

Mutable sequences
https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

The Python programming Language: **Tuples**

A Tuple is, basically, an immutable list:

```
a_tuple = (1, 2, 3)
str_tuple = ("a", "b", "a")
weird_tuple = ("1", 1, 1.0)

# access an element
a_tuple[2] # 3

# CANNOT set a value

# membership test
"1" in weird_tuple # True

# CANNOT add item to a tuple
# CANNOT remove item from a tuple

# find the index of the first specified element
a_tuple.index(2) # 1 -> a_tuple[1] is 2
```

Operations supported sequences https://docs.python.org/3.10/library/stdtypes.html#sequence-types-list-tuple-range

The Python programming Language: **Dictionaries**

A dictionary is an *unordered*, *changeable* and *indexed collection*.

Dictionaries can be created by placing a comma-separated list of `key:value` pairs within braces, or by the `dict` constructor.

reference
https://docs.python.org/3/library/stdtypes.html#mapping-types-dict

```python
a = {'one': 1, 'two': 2, 'three': 3}
b = dict(one=1, two=2, three=3)

# access an element
a['two'] # 2

# set a value
b['three'] = "3" # {'one': 1, 'two': 2, 'three': '3'}

# membership test
"one" in a # True

# add item to dict
a['four'] = 4

# remove item
a.pop("four")
del a["four"]
```

The Python programming Language: **Sets**

- A set is an *unordered collection* with *no duplicate* elements.
- Support to mathematical operations like union, intersection, difference, and symmetric difference.

- Curly braces or the set() function can be used to create sets. Empty set you have to use `set()`, not `{}`

reference

https://docs.python.org/3/tutorial/datastructures.html#sets

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership testing
True
>>> 'crabgrass' in basket
False
```

The Python programming Language: **Sets**

- A set is an *unordered collection* with *no duplicate* elements.
- Support to mathematical operations like union, intersection, difference, and symmetric difference.

- Curly braces or the set() function can be used to create sets. Empty set you have to use `set()`, not `{}`

reference

https://docs.python.org/3/tutorial/datastructures.html#sets

```
# Demonstrate set operations on unique letters from two words
>>>
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b # letters in both a and b
{'a', 'c'}
>>> a ^ b # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

The Python programming Language: **The `if` statement**

Usual logical predicates

Beware of the indentation!!!

```
if robot == 'rabbit':
    print('Hi, rabbit')
else:
    print('Hi, stranger')
```

```
# elif
if robot == 'rabbit':
    print('Hi, rabbit')
elif robot == 'hare':
    print('Hi, hare')
elif robot == 'turtle':
    print('Hi, turtle')
```

reference

https://docs.python.org/3/tutorial/controlflow.html#if-statements

The Python programming Language: **The `match` statement**

- A `match` statement takes an expression and compares its value to successive patterns given as one or more case blocks.
- This is superficially similar to a switch statement in C, Java or JavaScript (and many other languages), but *much* more powerful.

- Since Python 3.10

reference

https://docs.python.org/3/tutorial/controlflow.html#match-statements

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case 401 | 403 | 404:
            return "Not allowed"
        case _:
            return "Something's wrong with the Internet"
```

HTTP error codes

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

The Python programming Language: **The `match` statement**

- Patterns can look like unpacking assignments, and can be used to bind variables:

```
# point is an (x, y) tuple
match point:
    case (0, 0): # 1
        print("Origin")
    case (0, y): # 2
        print(f"Y={y}")
    case (x, 0): # 3
        print(f"X={x}")
    case (x, y): # 4
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

PEP 636 – Tutorial

https://peps.python.org/pep-0636

- 2 and 3 combine a literal and a variable, and the variable binds a value from the subject (`point`).
- 4 captures two values

The Python programming Language: `while` loops

Similar to C-derived languages

```
# Print i as long as i is less than 6
i = 1
while i < 6:
   print(i)
    i += 1

# break – stop looping if i is 3
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1

# continue – skip iteration for 5
i = 1
while i < 6:
    i += 1
    if i == 5:
        continue
    print(i)
```

Structured program theorem (Böhm-Jacopini)

Any algorithm can be expressed using just:

- Sequence
- Iteration
- Selection

The Python programming Language: `for` **loops**

More a *foreach* than a C-like *for*.

Iterate over a sequence (list, tuple, dictionary, string, etc.)

Range built-in function

https://docs.python.org/3/library/stdtypes.html#ranges

```
# over a list
fruits = ["apple", "pear", "orange"]

for fruit in fruits:
    print(fruit)

# break and continue like while

# range(start, stop[, step]) -> list of integers – iterate

# for a specified number of times
for i in range(10):
    print(i) # prints from 0 to 9
```

The Python programming Language: **functions**

```
def function_name(arguments):
    """docstring"""
    statement(s)
```

reference

https://docs.python.org/3/tutorial/controlflow.html#defining-functions

A function definition consists of the following components.

- Keyword `def` that marks the start of the function header.
- A function name to uniquely identify the function (same as identifiers).
- Arguments through which we pass values to a function (parameters). They are optional.
- A colon (:) to mark the end of the function header.
- Optional documentation string (`docstring`) to describe what the function does.
- One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
- An optional `return` statement to return a value from the function. `None` is returned when a `return` statement is missing.

The Python programming Language: **functions**

```python
def greet(name):
    """
    This function greets the person passed in as
    a parameter
    """
    print(f"Hello, {name}. Good morning!")

greet('Ugo') # Hello, Ugo. Good morning!
```

reference

https://docs.python.org/3/tutorial/controlflow.html#recap

Functions can have a number of mandatory arguments (positional) that have to be specified when calling the function.

Default values for arguments are allowed.

```python
def greet(name, msg="Good morning"):
    """
    This function greets the person passed in as
    a parameter with an optional message
    """
    print("Hello, {name}. {msg}!")

greet('Egidio') # Hello, Egidio. Good morning!

greet('Egidio', "Good Evening") # Hello, Egidio. Good Evening!
```

The Python programming Language: **functions**

Sometimes, we do not know in advance the number of arguments that will be passed into a function. We can handle this kind of situation through function calls with an arbitrary number of arguments.

In the function definition, use an asterisk (*) before the parameter name to denote this kind of argument.

```
def greet(*names):
    """This function greets all
    the person in the names tuple."""

    # names is a tuple with arguments
    for name in names:
        print("Hello", name)

greet("Ugo", "Stefano", "Egidio")
```

reference

https://docs.python.org/3/tutorial/controlflow.html#arbitrary-argument-lists

The Python programming Language: **Classes**

To create a class, use the keyword `class`

```
class Person:
    """This is a person"""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def introduce(self):
        print(f"Name: {self.name}, age: {self.age}")

p1 = Person("Joe", 26)

print(f"Name: {p1.name}, age: {p1.age}")

p1.introduce()
```

- Use `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created.

- Instance Methods are definded as functions with `self` as first parameter. `self` is a reference to the instance.

Object Oriented Programming (OOP)

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

https://en.wikipedia.org/wiki/Object-oriented_programming

The Python programming Language: **Modules**

A file containing a set of functions you want to include in your application.

To create a module just save the code you want in a file with the file extension `.py`

To use a module, it must be imported using the `import` statement:

reference

https://docs.python.org/3/reference/datamodel.html#modules

```
# assuming to have our previous greet function in the
# greeter_module.py file in the env var PYTHONPATH

import greeter_module

greeter_module.greet('Ugo")
```

The Python programming Language: **Exceptions**

Handle runtime errors programmatically.

A `try` statement followed by one or more `except` clause and an optional `finally` clause.

reference
https://docs.python.org/3/library/exceptions.html

reference
Built-in exceptions hierarchy

```
try:
    ... # code that can raise an instance of
SomeException

except SomeException as ex:
    ... # execute when SomeException raised in try
block

finally:
    ... # execute in any case
```

`raise` statement to raise an exception (built-in or custom)

```
raise ValueError("Invalid parameter")
```

The Python programming Language: **main function**

- When Python interpreter reads a source file, it will execute it line by line.
- When Python runs the "source file" as the main program, it sets the special variable (`__name__`) to have a value (`"__main__"`).
- When you execute the main function, it will then read the **`"if"`** statement and checks whether `__name__` does equal to `__main__`.
- In Python "`if __name__== "__main__"`" allows you to run the Python files either as **reusable modules or standalone programs.**

```
def myfun():
    print("Hello World!")

if __name__== "__main__":
    myfun()

print("other code")
```