SVILUPPO RISORSE PER L'AZIENDA
CISITA
PARMA

Unione Parmense degli Industriali

GIA Gruppo Imprese Artigiane

Introduction to Python

Internet of Things e Analisi predittiva

Gianfranco Lombardo MEng, Ph.D Candidate in ICT
gianfranco.lombardo@unipr.it

Lecture summary

- Python basics (about 1h)
  - Variables and data types
  - Control flow statements: if and while
  - Functions
  - Lists and tuples
  - Loops over sequences: For statement
- Exercises with practice (about 1h)
- 15 minutes break
- Python basics - part 2 (about 1h)
  - Files and data streams
  - Introduction to OOP in Python
- Exercises with practice (about 1h)

What is Python ?

Python is an interpreted, high-level, general-purpose programming language

Created by Guido van Rossum in 1991

Multi-paradigm (Structural, OOP and partially Functional)

Emphasizes code readability

It is dynamically typed

Most used language for Data Analysis and Machine Learning

What is Python ?

Installation

- Download and install the most recent version of Python (or at least version 3.6.6) : https://www.python.org/downloads/
- Editor : https://notepad-plus-plus.org/downloads/
    - Alternative - Jupyter notebook: https://jupyter.org/install (install with pip)

Data types

A type is an attribute of data which tells the interpreter how the programmer intends to use the data and which values and operations are allowed

Python provides **int, float, string, bool**

int: integer values (e.g, 5, 10, 1000…)
float: floating point values (e.g, 5.33, 3.14, 9.2 …)
string: single characters or words or sentences
bool: logic condition, True or False

Operations on data types

- **int and float**
  - Basic operations: **+ , - , * , /**
  - Comparison: **<, <= , >, >=, ==, !=**
- **bool**
  - Allowed operations: **and, or , not**
- **string**
  - More details later

```
>>> 4 + 5                  #Sum between two int
9                          #Result is an int again
>>> 3 / 2                  # Division between two int
1.5                        # Result is a float
>>> 5 < 3 or 4 == 5        # Logic condition
False                      # Result is a bool
```

Py

Variables and assignment

We need to store values to work with them

Think to a box (**variable**) where we can store values to be able of using them when we need

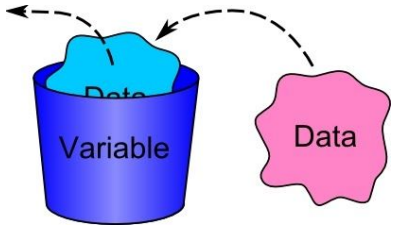
Variable's data type is automatically defined by the assigned value and is dynamic

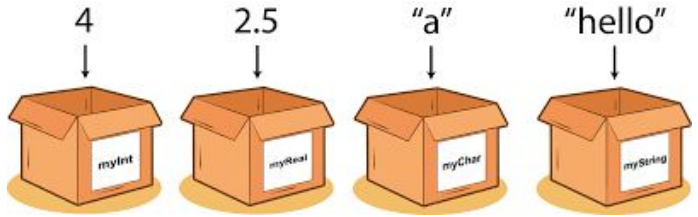
It is not a good practice to change data type of a variable

More suggested: use in case another variable

Operation allowed for each variable depends on its data type

Before using it we must define the variable with an assignment





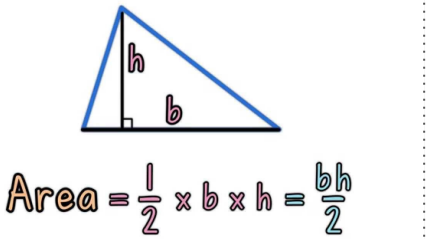
Assignment

variable_name = value

Assignment operator
Do not confuse with equality operator " ==" for logic expression

Py

```
>>> base = 5        #variable name is "base" and its value is 5, so base is an int variable
>>> height = 3.5    #variable name is "height" and its value is 3.5, so height is a float variable
>>> area = (base*height) / 2
>>> print(area)
8.75
#print function print on the screen the value of the variable inside the brackets
>>> print( type(area)) # print data type of variable area
<class 'float'>
```



Strings and text

A string variable can be a single character or a **list** of characters:

word = “a”

word = “Hello”

sentence = “My name is Mario Rossi”

Character can be also for example white spaces, “\n” for a new line or “\t” for tab

Different operations are allowed for strings, for example:

+ : concatenation

split(“,”) : To divide a string when a comma occurs

replace(“b”,”c”): to replace each “b” with a “c”

More things will be more clear when we will introduce the concept of **list**

Operations with strings

```
>>> name = "Mario"
>>> last_name="Rossi"
>>> complete_name= name+last_name
>>> print(complete_name)
MarioRossi
>>> complete_name = name+" "+last_name                    #concatenation
>>> print(complete_name)
Mario Rossi
>>> sentence= complete_name+" is walking in the park"
>>> print(sentence)
Mario Rossi is walking in the park
>>> print("\t"+sentence)
        Mario Rossi is walking in the park
>>> complete_name = complete_name.replace("o","a")     #replacement of each “o” with an “a”
>>> print(complete_name)
Maria Rassi
```

Input from keyboard

```
>>> name = input(" Your name is: ")
 Your name is: Mario
>>> last_name= input("Your last name is: ")
Your last name is: Rossi
>>> print(name+" "+last_name)
Mario Rossi
```

Py

- input() is a **function** that ask for an interaction with the keyboard and associates the received value in a variable
- Between the brackets is it possible to add a text message to be displayed during the interaction
- *WARNING*: Input returns always a string variable
  - If you want to use that values with another data type the variable must be **cast** with that data type

Cast of a variable

Change data type of a variable without re-assignment it
Dangerous but sometimes necessary

Typical error Py

```
>>> pi= 3.14
>>> radius = input("Insert the radius value ")
Insert the radius value 5
>>> area = pi*radius*radius                #Power can be also computed with the ** operator
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
```

Py

```
>>> pi= 3.14
>>> radius = input("Insert the radius value ")
Insert the radius value 5
>>> area = pi**float(radius)        #cast value of radius in a float data type
>>> print(area)
305.2447761824001
```

Pay attention! More checks required

Cast can be dangerous if we are not sure about the value inside the variable!

Typical error *Py*

```
>>> pi= 3.14
>>> radius = input("Insert the radius value ")
Insert the radius value Mario
>>> area = pi**float(radius)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'Mario'
```

Other cast functions are string() and int()

Conditional statements - IF

Statement if checks if a boolean condition is valid and in case it executes the next statements inside it
Otherwise it is possible to use the statements else or elif to execute different instructions when the first condition in the “if” is not valid
Instructions inside these statements MUST BE indented with tabs or spaces, not a choice it is mandatory in Python

Syntax

```
if boolean_condition :
        instructions
elif boolean_condition :
        instructions
else:
        instructions
```

Py

```
if area <100 :
        print(“Area is smaller than 100”)
elif area == 100 :
        print(“Area is equal to 100”)
else:
        print(“Area is greater than 100”)
```

Example: String comparison

```
# Ask for two strings and tell which is the biggest one or if they are equal
# Comparison among strings takes into account also words' order in the alphabet


first_word = input("Insert the first word: ")
second_word = input("Insert the second word: ")

if first_word < second_word:
        print(first_word+" is smaller than "+second_word)
elif first_word > second_word:
        print(first_word+" is greater than "+second_word)
else:
        print("words are equal")
```

Example: Select an option

```
# ask for two values and print a list of possible operations
first_number= float(input(“First number: “))
second_number=float(input(“Second number: “))
options= “\tDigit 1 to sum\n\tDigit 2 to subtract\n\tDigit 3 to multiply\n\tDigit 4 to divide”)
print(options)
choose=int(input(“”))

if choose == 1:
        print(first_number + second_number)
elif choose == 2:
        print(first_number - second_number)
elif choose == 3:
        print(first_number * second_number)
elif choose == 4:
        print(first_number /second_number)
else:
        print(“Wrong selection”)
```

Py

While loop statement

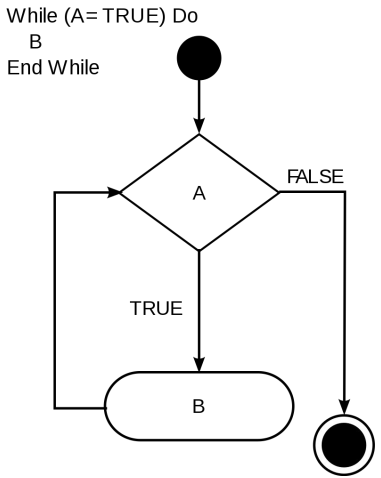
- While statement allows code to be executed repeatedly based on a given Boolean condition
  - It can be seen as “repeat until condition is false”
- Preliminary check for the boolean condition!
  - It is possible that the portion of the code inside will never be executed

Syntax

```
while boolean_condition :
        instructions
```

Py

```
i = input(“Insert a number below 100: “)
while float(i) <= 100:
        print(i**2)          #compute the square
        i = input(“Insert a number below 100: “)
```



Example: Select an option until correct choice

```
# ask for two values and print a list of possible operations
first_number= float(input(“First number: “))
second_number=float(input(“Second number: “))
option_correct = False
while not option_correct:
        options= "\tDigit 1 to sum\n\tDigit 2 to subtract\n\tDigit 3 to multiply\n\tDigit 4 to divide"
        print(options)
        choose=int(input(""))
        if choose == 1:
                print(first_number + second_number)
                option_correct=True
        elif choose == 2:
                print(first_number - second_number)
                option_correct=True
        elif choose == 3:
                print(first_number * second_number)
                option_correct=True
        elif choose == 4:
                print(first_number /second_number)
                option_correct=True
        else:
                print("Wrong selection")
```

Py

Example: Average of N numbers

```
n = int(input("How many values? "))
total = 0
count = 0

while count < n:
    val = int(input("Val? "))
    total += val                    # total = total + val
    count += 1                      # count = count + 1

if count != 0:
    print("The average is", total / count)
```

Functions

- Execution flow can become very complex and some parts could be used multiple times
  - Idea: Define blocks of code that perform particular operations that we can call and use when is requested
- Functions are exactly what we need
  - They have a name to be invoked
  - They can (optionally) take arguments as input
  - They can return an output (optionally)
- Usually also the main part of a program is inside a general function called “main”

Functions

Syntax

```
def function_name( arg1, arg2... ):
        instructions
        return var      #optionally
.
.
function_name(...) #invokation
```

Py

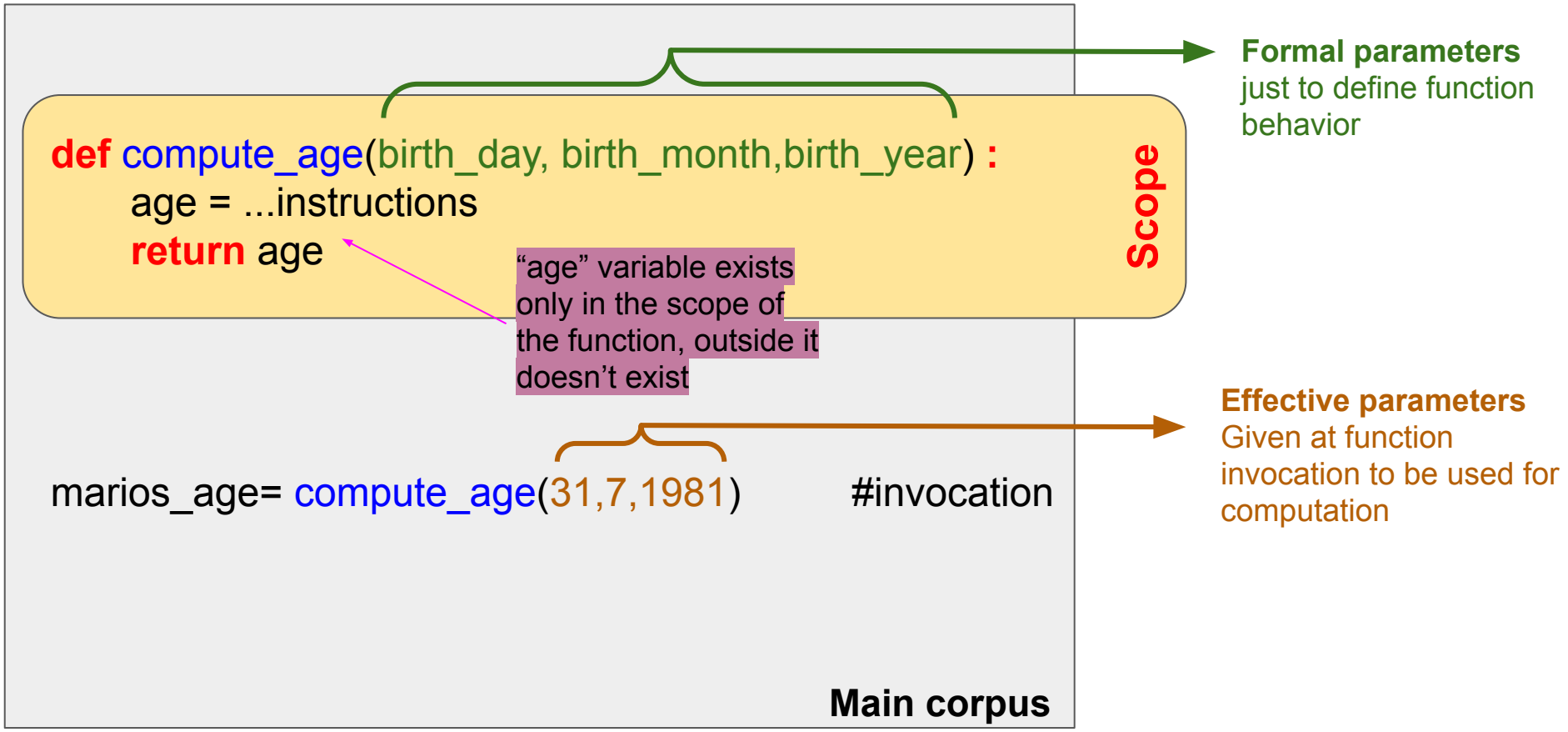
```
def hypotenuse( leg1, leg2 ):
        hyp = (leg1 ** 2 + leg2 ** 2) ** 0.5

        return hyp      .

hypotenuse(4,5) #invokation
```

- Indentation is still mandatory to define which sequences are inside the function and which not
- Functions must be defined before using it
- Inside a function can be everything (if, while, invocations to other functions and so on…)
- **Variables have a scope!** Variables defined in a function can be only used inside it and not outside
    - otherwise Global function, but it is not our topic for now

Functions


def compute_age(birth_day, birth_month,birth_year) :
age = ...instructions
return age
Scope
"age" variable exists only in the scope of the function, outside it doesn't exist
marios_age= compute_age(31,7,1981) #invocation
Main corpus
Formal parameters
just to define function behavior
Effective parameters
Given at function invocation to be used for computation

Example

```
def compute_area(s1,s2):
        a = s1*s2
        return a
def compute_perimeter(s1,s2):
        p = s1*2 + s2*2
        return p

def main ():
        #This is the main corpus of the program
        #ask rectangle's sides
        side1 = float(input("1: "))
        side2 = float(input("2: "))
        area=compute_area(side1,side2)              #invoke the function we have previously defined
        perimeter=compute_perimeter(side1,side2)
        print("Area is "+str(area)+" perimeter is "+str(perimeter))

main()                      # invocation of the main function so the execution can start
```

Lists (or vector)

- **Mutable** sequence of elements (or collection of objects)
  - think to a box with inside an ordered collection of other boxes
- They have a name like the variable but they have also a length to know how many elements are inside *---> len(X) function*
- Each element has an index to retrieve data
  - indexes go from *0 to len(X)-1*

Py

```
months= [“Jan”,”Feb”,”Mar”,”Apr”,”May”,”Jun”,”Jul”,”Aug”,”Sep”,”Oct”,”Nov”,”Dec”]
day_week = [“Mon”,”Tue”,”Wed”,”Thu”,”Fri”,”Sat”,”Sun”]

print( len(months))          #len function returns the length of a list
>>> 12
print( len(day_week))
>>> 7
print( day_week[3])          #invoce a list with index to get that element
>>> “Thu”
```

Operations with lists

- Attention! Use always a valid index 0 <= index <= len(X)-1
  - If index == -1, last item selected

Py

```
months=[“Jan”,”Feb”,”Mar”,”Apr”,”May”,”Jun”,”Jul”,”Aug”,”Sep”,”Oct”,”No
v”,”Dec”]
spring = months[2:5]          # ["Mar", "Apr", "May"]
quart1 = months[:3]           # ["Jan", "Feb", "Mar"]
quart4 = months[-3:]          # ["Oct", "Nov", "Dec"]
whole_year = months[:]        # Copy of the whole list
```

Insertion and removal

```
grocery = ["spam", "egg", "beans"]

grocery[0] = "sausage"       # replace an element

grocery.append("bacon")      # add an element to the end
grocery.pop()                # remove (and return) last element

grocery.insert(1, "bacon")   # other elements shift
removed = grocery.pop(1)     # remove (and return) element at index

if "egg" in grocery:         # True, grocery contains "egg"
        grocery.remove("egg")  # remove an element by value
```

Py

Equality and identity

```
a = ["spam", "egg", "beans"]
b = a[:]          # new list!
b == a            # True, they contain the same values
b is a            # False, they are two objects in memory
                  # (try and modify one of them...)
c = a
c is a            # True, same object in memory
                  # (try and modify one of them...)

d = ["sausage", "mushrooms"]
     grocery = c + d  # list concatenation --> new list!

a.reverse()       #reverse modify directly the list, no return
>>["beans","egg","spam"]
```

Strings: Now probably more clear

```
txt = "Monty Python's Flying Circus"
txt[3]    # 't'
txt[-2]   # 'u'
txt[6:12] # "Python"
txt[-6:]  # "Circus"

days = ["tue", "thu", "sat"]
txt = "|".join(days)  # "tue|thu|sat"

txt = "mon|wed|fri"
days = txt.split("|")  # ["mon", "wed", "fri"]

txt = “My name is Mario”
txt.lower()     #lower case
>>”my name is mario”
```

Tuple

- **Immutable** sequence of values, even of different type

Py

```
# Tuple packing
pt = 5, 6, "red"
pt[0]  # 5
pt[1]  # 6
pt[2]  # "red"

# multiple assignments, from a tuple
x, y, colour = pt  # sequence unpacking
a, b = 3, 4
a, b = b, a
```

Loops on sequences: FOR

- “FOR” statement is useful to iterate over collection of objects (lists,tuples,matrix..so on)

```
grocery = ["spam", "egg", "bacon"]
for product in grocery:
        print(product)
```

```
for val in range(0, 5):
        print(val * val)


    # 0 1 4 9 16
```

- At each iteration, a different element of grocery is assigned to product
- range: range of consecutive values
  - Lower end included (defaults to 0)
  - Upper end excluded

Exercise: Repetita iuvant

- Try to write again (without copy&paste) the previous example ***“Example: Select an option”***
- Tell to the user if his selection is wrong until it is not correct!
- Then try to modify in the following way:
  - Option 1: Compute the area and perimeter of a rectangle
  - Option 2: Compute the area of a triangle
  - Option 3: Sum both values to have the diameter, compute the radius and then the area of a circle
- Print the correct results

Exercise: Cryptography

- Define two lists:
  - plain_chars =["a","b","c","d","e","f","g","h","i","l","m","n","o","p","q","r","s","t","u","v","z"]
  - crypto_chars = list(range(0,21)) #returns a list with number between 0 and 20
  - crypto_chars.reverse() #reverse
- Ask the user for three sentences he wants to keep safe
  - Convert sentences in lower case
  - Encrypt these sentences by replacing each char in plain_chars (if exist) with the correspondent one in crypto_chars
  - Print the three sentences encrypted

My name is Mario

10y 9201016 124 10205128

Exercise: Registry

- Ask to insert a name and lastname in a single input request and the date of birth in another one, until the user insert the special character “0” to finish
- Like: >>> Insert name and lastname: Mario Rossi
- >>> Insert the age of Mario Rossi: 34

    - Save name and lastname in different lists (split the string before :))
    - Compute and save the age in another list
        - These three lists are aligned by the index
    - Print each person with his age in a single row
        - like: “Mario Rossi 34”
    - Delete from the register people with an age above 65
    - Print the remaining people again

BREAK

Files

- In order to work with files we have to open a stream using particular functions
  - This stream will be linked to an object and will be stored in a variable

- Files can be divided into two main categories:
  - **Text files** (.txt, .csv...etc)
    - Plain text usually encoded with UTF-8 format, ANSI or UCS
  - **Binary files**
    - Encoding depends on the application source
    - Private format (.docx)
    - Public format (.jpeg and others)

- In this course we will work only on text files

Files

- To open our data stream we have to define
  - File name and its path
  - Rules to open the file
    - “r” : read-only
    - “w”: write
    - “a”: write in append
    - “w+”,”a+”: write o write append and create the file if it doesn’t exist
    - Mixture of these rules

```
f1 = open(“my_file.txt”, “r”)     # f1 is our file stream and we can only read the content of my_file.txt
f1.readline()                     # Reads one line of the file until the \n character occurs
f1.readlines()                    # Reads all lines and stores in a list of strings
f1.close()                        # When finished, close the file
```

“with open” pattern

- The most used way of working with files in Python is using the “with open” pattern
- It is like defining a function and the data stream has its scope only inside that block of codes
- It is not necessary to close the stream! More efficient.

```
with open(“my_file.txt”, “r”)  as f1:
        for line in f1.readlines():
                print(line)
                fields= line.split(“,”)        #divide each line by comma, it returns a list of strings
                print(fields)
```

“with open” pattern

- When we want to write, remember to cast everything as a string

```
with open(“my_file.txt”, “w+”)  as f1:
        for value in range(0,10):
                f1.write(str(value)+”\n”)       #Write one number for each row
```

Example A:

```
ask=True
f1 = open("products_list.txt","a+")
while(ask):
        request = input("insert a product comma and its price or end to finish:\n")
        if request != "end":
                if "," in request:                      #check if comma is present
                        fields= request.split(",")
                        product=fields[0]
                        price=float(fields[1])
                        f1.write(product+","+str(price)+"\n")       #write on file
                else:
                        print("bad insertion!")
        elif request == "end":
                ask=False
                print("Goodbye!")
```

Example B:

```
products_list= []                #initialize a list that will contain the products

with open("products_list.txt","r") as f1:
        for line in f1.readlines():
                fields= line.replace("\n","").split(",") #Replace line end character and split by comma
                if len(fields)>1:
                        item= fields[0]
                        products_list.append(item)
                        price= float(fields[1])
                        print("Product is "+item+" and its prices is "+str(price))
        print(products_list)        #print all products in the list
```

Py

Object Oriented Programming

- Object Oriented Programming (OOP) is a programming paradigm based on the concept of "objects"
- An object contains data in the form of fields (called attributes)
- At the same time it is something more complex than just a variable because it offers services by providing functions to manage data inside (methods)
- They have an internal state

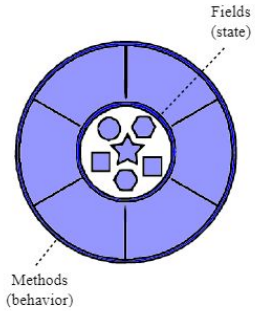
- For now, we can think to objects as the initialization of a custom data type that someone have defined previously

- **The big novel**: Everything in Python is an object and you already used objects in our previous exercises !

OOP: What is an object?

- Represents a concept or a way to model physical objects (e.g, a sensor, a product, a character in a video game! etc..)
- Example:
  - object name: “Mario’s car”
  - class : car #We will define it soon
  - attributes:
    - 4 wheels
    - Speed
    - Current gear
  - methods:
    - acceleration()
    - drive()
    - change gear()

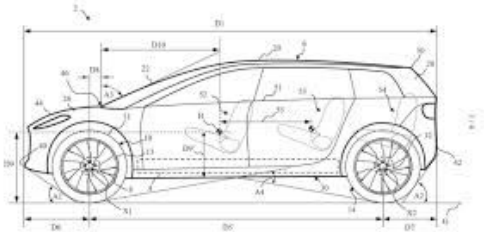
OOP: What is a class ?

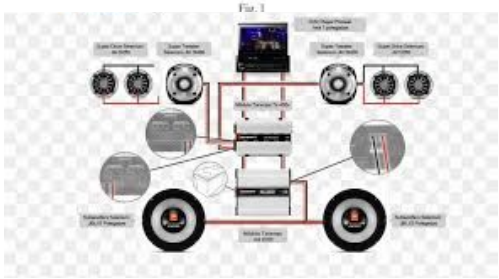


- A class defines the structure of an object:
  - What are the attributes type and name
  - What are the methods that the object has to offer
  - What is the code that methods should execute
- Each object has a class that define its characteristics
- An object is the concrete form of a class!
- Example: **class** of soccer teams, each soccer teams has 11 players etc..
  - **Objects** of type “soccer teams”: Juventus, Milan, Inter, Parma...so on
- Example: **class** car
  - **Objects**: mario’s car, alice’s car, george’s car

OOP: Class Vs Objects

- Class
  - What is the concept?
  - How does it work?
  - How is it build?





- Object
  - Concrete instance that we can use
  - It is built depending on the rules that its class defines

Everything is an object

- You already used objects in Python!
- Everything is implemented as an object!
- When you define:
  - value=5 an object called “value” is created from class “Integer” that defines what is an int value and what methods we can use on it to change its internal state
  - products_list=[] an object from class list is created
  - message = “Hello world” an object from class String is created

- In fact, on these objects for example strings we called some functions to modify their internal state
  - .split()
  - .replace()
  - .append() for lists

OOP: Class in Python

- Each class should define:
  - The internal state of an object with attributes (read variable under control of each object)
  - Methods to modify the internal state (read functions)
  - A special function called “constructor” that permits to build an object of that class
    - It takes arguments (optionally) to define the initial internal state and return an instance of an object of that class type

Define a class

```
class Product:
        def __init__(self, name,price,number):
                self._name = name               # attribute
                self._price = price             # attribute
                self._number = number           # attribute

        def get_name(self):                     #method
                return self._name
        def get_price(self):                    #method
                return self._price
        def set_price(self,new_price):          #method
                self._price=new_price
def main():
        p = Product("pen",1.50,200)             #Instantiation of an object p (Product type)
        print(p.get_name())                     #call of a method on object P
        print(p.get_price())
        p.set_price(2.00)
        print(p.get_price())
main()
```

Py

self: special keyword that say to interpreter that methods or attributes value have to be assigned to each object that is built from this class !

Special objects with special classes

- Now more things about Python should be more clear (or at least I hope :D)
- Everything is an object so everything refers to particular classes!
- For example: integer values are instances (object) of class Integer
    - For this primitive class (built-in in Python) special constructors functions are defined
        - We say n=5 but in theory we can also say n = int(5)
            - The way we were doing variable casting!
- So remember when you use “built-in” data type you can call the available methods on these objects!
- You can also define your custom classes and instantiate your own objects!
- You can combine these things: For example a list of your custom objects!
- You can use classes defined by somebody else!
    - Software library or API

Define a class

```
class Product:
        def __init__(self, name,price,number):
                self._name = name               # attribute
                self._price = price             # attribute
                self._number = number           # attribute

        def get_name(self):                     #method
                return self._name
        def get_price(self):                    #method
                return self._price
        def set_price(self,new_price):          #method
                self._price=new_price
def main():
        p = Product("pen",1.50,200)             #Instantiation of an object p (Product type)
        print(p.get_name())                     #call of a method on object P
        print(p.get_price())
        p.set_price(2.00)
        print(p.get_price())
main()
```

List of objects

Py

```
n = 0
products_list=[]
while n<10:
        request = input(“Insert product,price,quantity”)
        fields= request.split(“,”)
        p = Product(fields[0],float(fields[1]),float(fields[2]))
        products_list.append(p)
        n+=1
for item in products_list:
        p.set_price(p.get_price()+2)
        p.buy() # this method could reduce the number of available items ! Has to be defined in the class
```

Modify product class

```
class Product:
        ……..
        def get_name(self):                     #method
                return self._name
        def get_price(self):                    #method
                return self._price
        def set_price(self,new_price):          #method
                self._price=new_price


        def buy(self):
                if self._number >0 :
                        self._number-=1
```

Exercise: Registry with OOP and files

- Starting from previous Registry exercise
  - Define a class Person with attributes name, last name and age
  - Define methods you will need to set these attributes and to compute the age
  - Define a list of objects Person and put each item in this list
  - After 10 items, iterate over the list and write all information on a file “registry.txt”
    - One person for each row

Exercise: Registry with OOP and files

- Read registry.txt, create an object for each person and add to a list
- Compute the average age in the registry by defining an apposite function

- Try to order your list depending on the age
  - Advice: Use a second list and add items directly ordered by iterating over the first list
- After that, write on a file again people ordered using “append” mode on the same registry.txt file