Computing Systems
An introduction to Python

Andrea Masciadri, PhD
andrea.masciadri@polimi.it

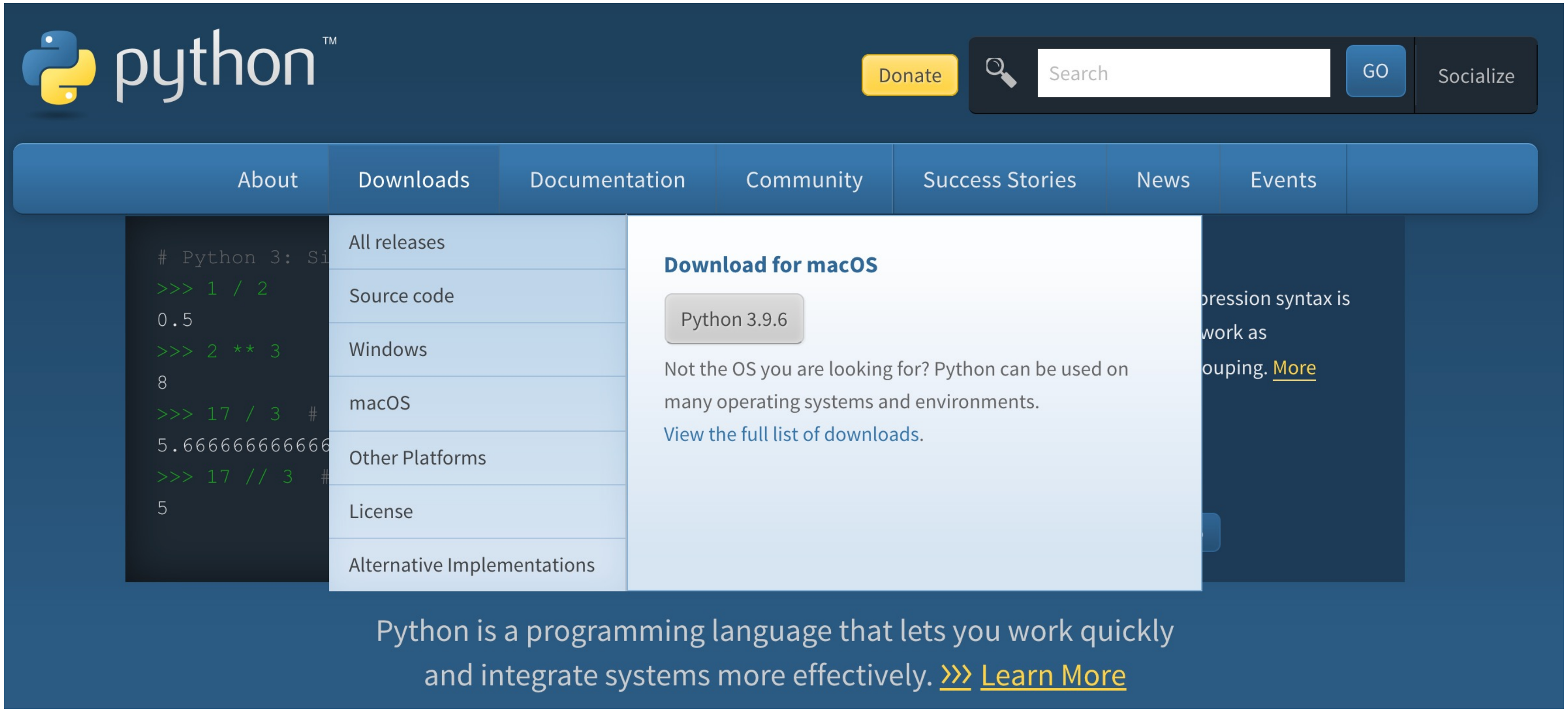
Agenda

- Introduction
- Data structures
- Syntax

INTRODUCTION

Python Installation

www.python.org



```
# Python 3: Si
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3  #
5.66666666666
>>> 17 // 3  #
5
```

Download for macOS

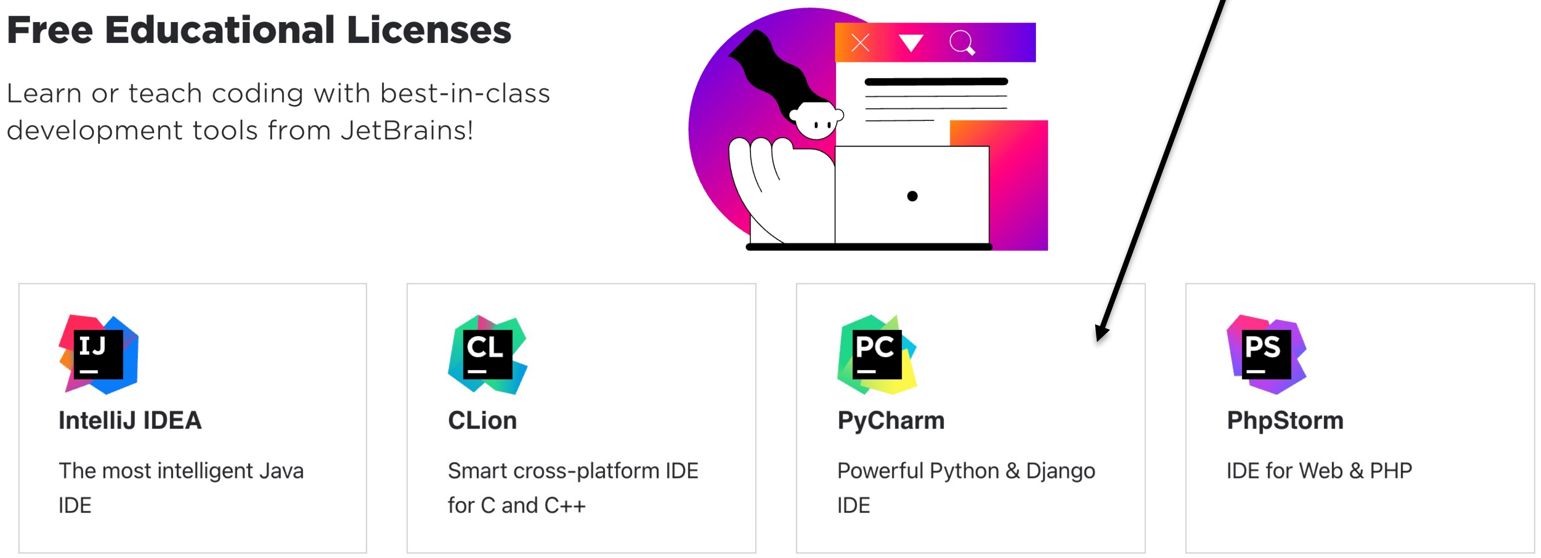
Not the OS you are looking for? Python can be used on many operating systems and environments.
View the full list of downloads.

pression syntax is work as ouping. More

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

IDE Installation

PyCharm is available for students under a Free Educational Licence
https://www.jetbrains.com



Free Educational Licenses

Learn or teach coding with best-in-class development tools from JetBrains!

IntelliJ IDEA

The most intelligent Java IDE

CLion

Smart cross-platform IDE for C and C++

PyCharm

Powerful Python & Django IDE

PhpStorm

IDE for Web & PHP

Python VS C

	C	Python
Architecture	Procedural, high-level, general-purpose, compiled programming language	multi-paradigm, (also) interpreted programming language
Variables	requires a compulsory declaration of variable types	loosely-typed Dynamically-typed
Pointers	yes	not supported
Garbage collector	not supported	yes
Indentation	Nice have	mandatory
Built-in functions	limited number of built-in functions	large library of built-in functions

Object oriented programming (OOP)

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

Wikipedia

Class

Is a template that defines the behavior of an object, defines the data (attributes) and the functions that operates on the data (methods).

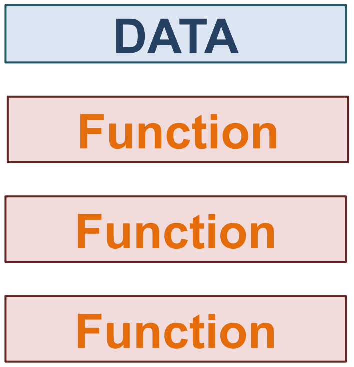
Object

Is an instance of a class, initialized with specific data, objects are created and eventually destroyed at run time and belongs to a specific class.

OOP Core principles

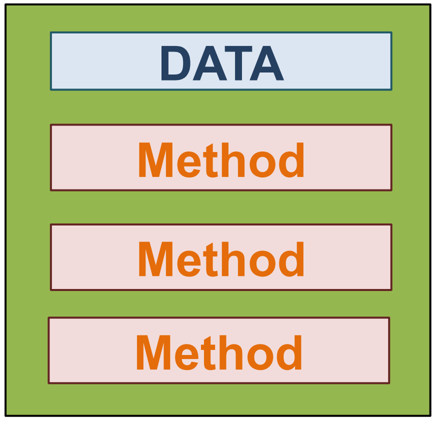
Encapsulation: Bind the data with the code that manipulates it

Procedural



•Data is accessible to all the functions of the program

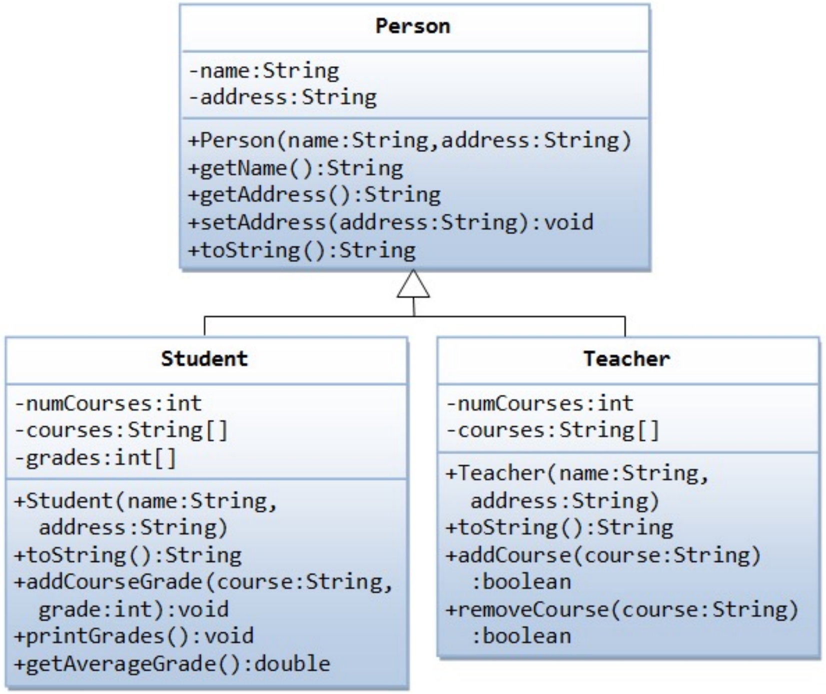
Object Oriented



- Data is encapsulated in the object invisible to the world "outside"
- Methods can be used to access and manipulate data

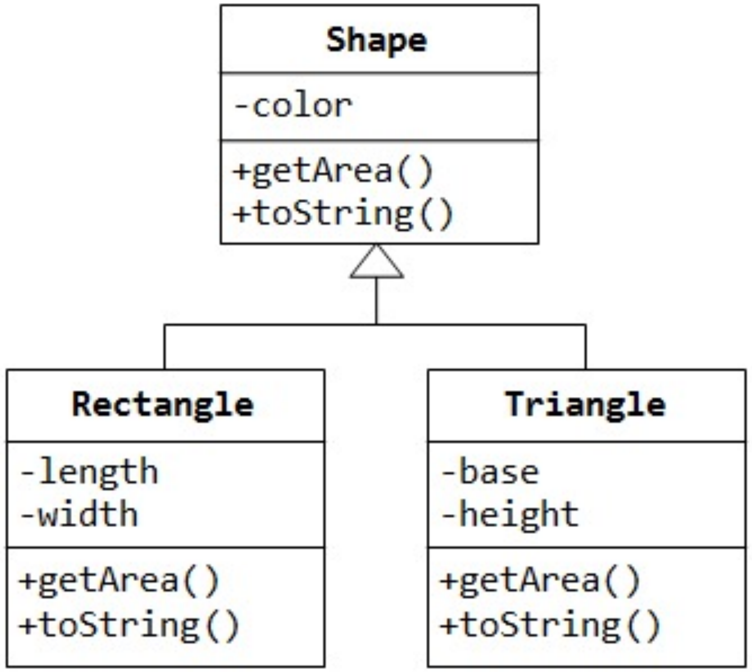
OOP Core principles

Inhetitance: A child object can extend the behavior of a paren object


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
Student
-numCourses:int
-courses:String[]
-grades:int[]
+Student(name:String, address:String)
+toString():String
+addCourseGrade(course:String, grade:int):void
+printGrades():void
+getAverageGrade():double
Teacher
-numCourses:int
-courses:String[]
+Teacher(name:String, address:String)
+toString():String
+addCourse(course:String) :boolean
+removeCourse(course:String) :boolean

OOP Core principles

Polymorphism: Ability to present the same interface for different undelying form



Class example

```
# class definition
class Animal(object):
    # constructor of the class
    def __init__(self, n):
        self.name = n

    # a method
    def get_name(self):
        return(self.name)

### instances of class
lion = Animal('lion')
cat = Animal('cat')

animals = [lion, cat]
for a in animals:
    print(a.get_name())
```

```
lion
cat
```

Indentation

```
/* C code */
C = 0;
if (A > 6) {
    if (B < 3) {
        C = A + B;
    } else {
        C = A - B;
    }
}
```

INPUT:

A = 7   B = 2
A = 7   B = 4

A = 4   ...

OUTPUT:

C = 9
C = 3

C = 0

EQUAL!

```
/* C code */
C = 0;
if (A > 6) {
    if (B < 3) {
        C = A + B;
}
else {
        C = A - B;
}}
```

INPUT:

A = 7   B = 2
A = 7   B = 4

A = 4   ...

OUTPUT:

C = 9
C = 3

C = 0

Indentation

```
# Python code
C = 0
if A > 6:
    if B < 3:
        C = A + B
    else:
        C = A - B
```

INPUT:

A = 7 B = 2
A = 7 B = 4

A = 4 …

OUTPUT:

C = 9
C = 3

C = 0

DIFFERENT!

```
# Python code
C = 0
if A > 6:
    if B < 3:
        C = A + B
else:
    C = A - B
```

INPUT:

A = 7 B = 2
A = 7 B = 4

A = 4 …

OUTPUT:

C = 9
C = 0

C = 3

Data types

Python is a dynamic language but it is also strongly typed.
The interpreter keeps track of all variable types

```
>>> my_number = 123
>>> type(my_number)
<class 'int'>

>>> my_string = "a string"
>>> type(my_string)
<class 'str'>

>>> print(my_string + my_number)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Data types

How big can a python integer be?

```
1044388881413152506691752710716624382579964249047383780384233483283 9
53907971557456848826811934997558340890106714439262837987573438185793
60726323608785136527794595697654370999834036159013438371831442807001
18559462263763188393977127456723346843445866174968079087058037040712
84048740118609114467977783598029006686938976881787785946905630190260
94059957945343282346930302669644305902501597239986771421554169383555
988529148631823                                      413490084170616
750936683338505                                      213796825837188
091833656751221                                      259567449219461
702380650591324                                      982023131690176
780066751954850799216364193702853751247840149071591335459982790513399
61155179427110683113409058427288427979155484978295432353451706522326
90613949059876930021229633956877828789484406160074129456749198230505
71642377154816321380631045902916136926708342856440730447899971901781
46576347322385026725305989979599609079946920177462481771844986745565
92501783290704731194331655508075682218465717463732968849128195203174
57002440926616910874148385078411929804522981857338977648103126085903
00130241346718972667321649151113160292078173803343609024380470834040
3154190336
```

There is no limit

Packages and modules

Modules

- python files with .py extension that implements functions or classes
- imported into the code using **`import`** command
- python provides a set of built-in modules

Packages

- namespaces wich contain multiple package or modules
- implemented asa directory containing modules or other package
- the directory MUST contain a file called __init__.py that can be empty

Packages and modules

```
# Imports datetime module into current namespace
import datetime
today = datetime.date.today()
yesterday = today - datetime.timedelta(days=1)
print(today)
print(yesterday)


# imports datetime and add date and timedelta
# into current namespace
from datetime import date, timedelta
today = date.today()
yesterday = today - timedelta(days=1)
print(today)
print(yesterday)
```

Frameworks

- Stand alone applications
  - Console
  - Tkinter (Tcl,Tk)
  - PyQT
  - wxPython
- Web application
  - Flask microframework
  - Django
- Numerical analysis
  - NumPy
  - SciPy

Extensions

PIP is the racommended tool to install python packages

```
pip install django
pip install
git+git://github.com/django/django.git#egg=django
```

Virtual envirnments

- a program that separates environments in order to isolate dependencies for different projects.

DATA STRUCTURES

Boolean

```
# simple boolean
is_python = True

# Everything can be converted to boolean
is_python = bool("yes sure!")

# some things are equivalent to False
these_are_false = False or 0 or "" or {} or [] or None

# others are True
these_are_true = True and 1 and 2 and "Some Text" and
{'foo': 'bar'} and [1, 1, 2, 3, 5, 8]
```

Numbers

```
# Integers
year = 2016
year = int("2016")

# Floating point
pi = 3.14159265
pi = float("3.14159265")

# Fixed Point
from decimal import Decimal
price = Decimal("0.02")
```

List

```
# initializing list
empty_list = []
my_list = [1,5,10]
my_heterogeneous_list = [1, "foo", "bar", True]
nested_list = [1, [1, 10]]

#accessing list elements
len(my_list)          # => 3
print(my_list[0])        # => 1
print(my_list[0:2])      # => [1,5]
print(my_list[1:])       # => [5,10]

# adding elements
mylist.append(42)
mylist.extend(['python',"rulez", True])
```

Dictionary

```
person = {
    'name': 'John',
    'surname': "Doe"
}
person['age'] = 25                  # add field
print(person['name'])               # => John

>>> person.keys()
dict_keys(['age', 'name', 'surname'])

>>> person.values()
dict_values([25, 'John','Doe'])

>>> person.items()
dict_items([('age', 25), ('name', 'John'), ('surname',
'Doe')])
```

Strings

```
name = "I'm a string"

me_too = 'I am also a string using "am" instead of "m"'

multiline = """And I am
a multiline string that is
splitted on more than one line"""

multiline2 = '''also with
single quotes'''
```

SYNTAX

Comments

```
# inline comment

"""
This is a multiline comment that can be used to
create automatic documentation for code
"""

'''
Single quote can also be used instead double one
'''
```

Arithmetic operators

```
a = 10          # 10
a += 1          # 11
a -= 1          # 10

b = a+1         # 11
c = a-1         # 9

d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100
```

Other operators

- Logical operators
  - logical AND `a and b`
  - logical OR `a or b`
  - negation `not(a)`
- Aritmetic comparison
  - Ordering `> >= < <=`
  - Equality `==`
  - Difference `!=`

String manipulation

```
fullname = "John" + " " + "Doe"         # John Doe
fullname += " is my name"               # John Doe is my name

fullname = " ".join(["John","Doe","is my name"])

# this will give "Dec 31 1989"
my_string_date = '%s %d %d' % ('Dec', 31, 1989)

# this will give "John Doe is 27 years old"
my_label = '%(first)s %(last)s is %(age)d years old'
    % {'first': 'John', 'last': 'Doe', 'age': 27}
```

Conditionals

```
temperature = 22

if temperature < 15:
    print("cold")
elif temperature >=16 and temperature <25:
    print("warm")
else:
    print("cold")
```

Loops

for loop

```
fruits = ['apple', 'banana', 'kiwi']
for f in fruits:
    print(f)
```

```
apple
banana
kiwi
```

```
primes = [2, 3, 5, 7]
for n in primes:
    print(n)
```

```
2
3
5
7
```

```
misc = [1, '1', 'joe']
for m in misc:
    print(m, end="")
    print(": ", end="")
    print(type(m))
```

```
1: <class 'int'>
1: <class 'str'>
joe: <class 'str'>
```

```
for i in range(0,3):
    print(f)
```

```
0
1
2
```

Loops

for loop with dictionaries

```
persons = {
     'Andrea': 21,
     'Fabio': 22,
     'Simone': 31
}
for key, value in persons.items():
    print("%s, %s" % (key, value))
```

```
Andrea, 21
Fabio, 22
Simone, 31
```

while loop

```
x = 0
while x < 3:
     print(x)
     x += 1
```

```
0
1
2
```

Functions

Function definition

```
def print_welcome():
    print("Welcome to our powerful program.")
    print("Type X to exit or C to continue")
```

```
def adder(n1, n2):
    result = n1 + n2
    return(result)
```

Function use

```
print_welcome()
```

```
c = adder(12, 40)
```

Functions

Default value for parameters

```
def hello(message="Hello World"):
    print(message)

hello()                         # Hello World
hello("Ciao Mondo!")            # Ciao Mondo!
```

Positional parameters and keyword

```
def hello(how_many, message="Hello World"):
    print(hoe_many*message)

hello(1)                        # Hello World
hello(2, message="Ciao Mondo!")
                                # Ciao Mondo!Ciao Mondo!
```

References

- Fabio Salice, Simone Mangano - Python programming language.

POLITECNICO
MILANO 1863

ITECNICO
LANO 1863

Questions?

andrea.masciadri@polimi.it