Python

BASICS

Introduction to Python programming, basic concepts: formatting, naming conventions, variables, etc.

Identikit

- First appeared in 1991
- Designed by Guido van Rossum
- General purpose
- High level
- Emphasis on code readability and conciseness
- Website
  - http://www.python.org
- We will use Python 3
  - <u>not</u> Python 2

```c
#include <stdio.h>

int main()
{
    printf("Hello, world!");
    return 0;
}
```

```python
print("Hello, world!")
```

inline comment

```
# this will print "Hello, world!"
print("Hello, world!")
```

Keywords

- and
- del
- from
- not
- while
- as
- elif
- global
- or
- with

- assert
- else
- if
- pass
- yield
- break
- except
- import
- class
- exec

- in
- raise
- continue
- finally
- is
- return
- def
- for
- lambda
- try

Variables

this is a string

```
language_name = "Python"
```

naming convention: joined_lower

Python basics

Variables

```
language_name = "Python"
version = '3.7.0'
introduced = 1991
is_awesome = True
```

Type Inference

```
language_name = "Python" # string
version = '3.7.0'    # another string
introduced = 1991    # integer
is_awesome = True    # boolean
```

actual type can be checked with `type()`

Python basics

String

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

another_string = 'I'm a string, too'
```

String

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

another_string = 'I'm a string, too'
# SyntaxError: invalid syntax
```

String

```
another_string = 'I am a string, too'

another_strig = 'I\'m a string, too'
```

escape sequence

String

```
long_string = """I am a long string.
I span over two lines."""

long_string = '''I am another long
string.
I span over three lines.
I am composed by three sentences.'''
```

If Statement

```
people = 20
cats = 30

if people < cats:
    print("Too many cats! We are doomed!")
4 spaces


if people > cats:
    print("Not many cats! We are safe!")
4 spaces
```

If Statement

```python
people = 20
cats = 30

if people < cats:
    print("Too many cats! We are doomed!")
elif people > cats:
    print("Not many cats! We are safe!")
else:
    print("We can't decide.")
```

Comparators and Booleans Operators

```python
print(2 == 1)

print('string' == "string")

print(not False)

print(2==1 and True)

print(2==1 or True)
```

Comparators and Booleans Operators

```python
print(2 == 1) # False

print('string' == "string") # True

print(not False) # True

print(2==1 and True) # False

print(2==1 or True) # True
```

Characters

```
for char in "hello":
    print(char)
```

```
h
e
l
l
o
```

Python basics

Characters

```
say_hello = "hello!"
print(say_hello[1])
```

index

e

Characters

```python
say_hello = "hello!"
print(type(say_hello[1]))
```

```
<class 'str'>
```

Combining Strings

```
language_name = "Python"
version = '3.7.0'

python_version = language_name + version
# python_version is Python3.7.0

print("my " + "name") # my name
```

concatenation

Combining Strings

repetition

```
language_name = "Python"

a_lot_of_python = language_name*3
# a_lot_of_python is PythonPythonPython
```

Building Complex Strings

```
a = 3
b = 5

# 3 times 5 is 15
print(a, "times", b, "is", a*b)
```

works with `print()`, only

Building Complex Strings

```
a = 3
b = 5

# 3 times 5 is 15
result = a + " times " + b + " is " + a*b
```

Building Complex Strings

```
a = 3
b = 5

# 3 times 5 is 15
result = a + " times " + b + " is " + a*b
#TypeError: unsupported operand type(s)
```

Building Complex Strings

```
a = 3
b = 5

# 3 times 5 is 15
result = str(a) + " times " + str(b) + "
is " + str(a*b)
```

String Interpolation

```
a = 3
b = 5

# 3 times 5 is 15
result = "%d times %d is %d" % (a, b, a*b)
```

Specifiers
- %s, format strings
- %d, format numbers
- %r, raw representation

tuple

→ specifiers.py

String Interpolation

```python
a = 3
b = 5

# 3 times 5 is 15
result = "{} times {} is {}".format(a, b,
a*b)
```

new way!

String Immutability

```python
# hello
say_hello = "helko"


# ops…
say_hello[3] = "l"
```

String Immutability

```
# hello
say_hello = "helko"

# ops…
say_hello[3] = "l"
# TypeError
```

String Immutability

```
# hello
say_hello = "helko"


# ops…
say_hello = "hello"
```

Other operations with strings? ⟶ Python docs

Getting Input

```python
print("How old are you?")
age = input() # age is a string

print("You are " + age + " years old")
```

Getting Input

```python
print("How old are you?")
age = input() # age is a string

print("You are " + age + " years old")

# I want "age" to be a number!
age = int(input())
```

Python basics

Getting Input

```python
age = input("How old are you? ")

print("You are " + age + " years old")
```

List

```
fruits = ["apples", "oranges", "pears"]
count = [1, 2, 3, 4, 5]
change = [1, "pennies", 2, "dimes"]
```

a datatype to store multiple items, in sequence

Dictionary

key, immutable

value

```
legs = {"ant": 6, "snake": 0, "cow": 4}
states = {"Italy": "IT", "Germany": "DE"}
```

a datatype to store multiple items, not in sequence

Loops

```python
doctor = 1

while doctor <= 13:
    exterminate(doctor)
    doctor += 1
```

For Loop: Strings

```python
for char in "hello":
    print(char)
```

```
h
e
l
l
o
```

For Loop: Ranges

```python
for number in range(0,5):
    print(number)
```

```
0
1
2
3
4
```

For Loop: Ranges

```python
for number in range(0,25,5):
    print(number)
```

```
0
5
10
15
20
```

For Loop: Lists

```python
fruits = ["apples", "oranges", "pears"]

for fruit in fruits:
    print("I love", fruit)
```

```
I love apples
I love oranges
I love pears
```

For Loop: Dictionaries

```python
legs = {"ant": 6, "snake": 0, "cow": 4}

for (animal, number) in legs.items():
    print("{} has {} legs".format(animal, number))
```

```
ant has 6 legs
snake has 0 legs
cow has 4 legs
```

Printing a List

```python
to_buy = ["eggs", "milk"]
print(to_buy)
```

```
['eggs', 'milk']
```

Printing a List

```
to_buy = ["eggs", "milk"]
print(to_buy[0])
```

```
eggs
```

Modifying a List

```python
to_buy = ["eggs", "milk"]
print(to_buy[0])


to_buy[0] = "butter"
print(to_buy[0])
```

```
eggs

butter
```

Modifying a List

```python
to_buy = ["eggs", "milk"]

# I need to buy chocolate!
to_buy.append("chocolate")
```

```
['eggs', 'milk', 'chocolate']
```

Modifying a List

```python
to_buy = ["eggs", "milk"]
to_buy.append("chocolate")


to_buy.extend(["flour", "cheese"])
```

```
['eggs', 'milk', 'chocolate', 'flour', 'cheese']
```

Modifying a List

```
to_buy = ["eggs", "milk"]
to_buy.append("chocolate")
```

concatenation

```
to_buy = to_buy + ["flour", "cheese"]
```

```
['eggs', 'milk', 'chocolate', 'flour', 'cheese']
```
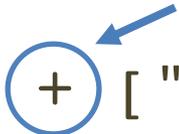
Modifying a List

```
to_buy = ["eggs", "milk", "chocolate",
"flour", "cheese"]
```

slice operator

```
print(to_buy[1:3])
```

```
['milk', 'chocolate']
```
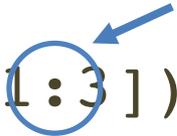
Modifying a List

```python
to_buy = ["eggs", "milk", "chocolate",
"flour", "cheese"]

# make a full copy of the list
remember = to_buy[:]
```

works with strings, too
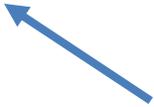
Modifying a List

```python
to_buy = ["eggs", "milk", "chocolate",
"flour", "cheese"]

# I don't need cheese!
to_buy.pop()

# … neither milk, by the way!
to_buy.pop(1)
```

Modifying a List

```python
to_buy = ["eggs", "milk", "chocolate",
"flour", "cheese"]

# I don't need cheese!
to_buy.remove("cheese")

# … neither milk, by the way!
to_buy.remove("milk")
```

Modifying a List

```python
to_buy = ["eggs", "milk", "chocolate",
"flour", "cheese"]


# I want my original list back!
del to_buy[2:6]
```

```
['eggs', 'milk']
```

Strings vs. Lists

A string is a sequence of characters…

　　　　　　… but a list of characters is <u>not</u> a string

```
language_name = "Python"

# string to list
name = list(language_name)
```

Strings vs. Lists

```python
sentence = "this is AmI"

# break a string into separate words
words = sentence.split()
```

```
['this', 'is', 'AmI']
```

Copying Lists

```python
fruits = ['apple', 'orange']
favorite_fruits = fruits

# add a fruit to the original list
fruits.append('banana')

print('The fruits now are:', fruits)
print('My favorite fruits are', favorite_fruits)
```

```
Fruits are: ['apple', 'orange', 'banana']
My favorite fruits are: ['apple', 'orange',
'banana']
```

???

Copying Lists

```python
fruits = ['apple', 'orange']
favorite_fruits = fruits

# add a fruit to the original
fruits.append('banana')

print('The fruits now are:', fruits)
print('My favorite fruits are', favorite_fruits)
```

We **do not** make a copy of the entire list, but we only make a **reference** to it!

Frames | Objects

Global frame
fruits
favorite_fruits

list
0 "apple" 1 "orange" 2 "banana"
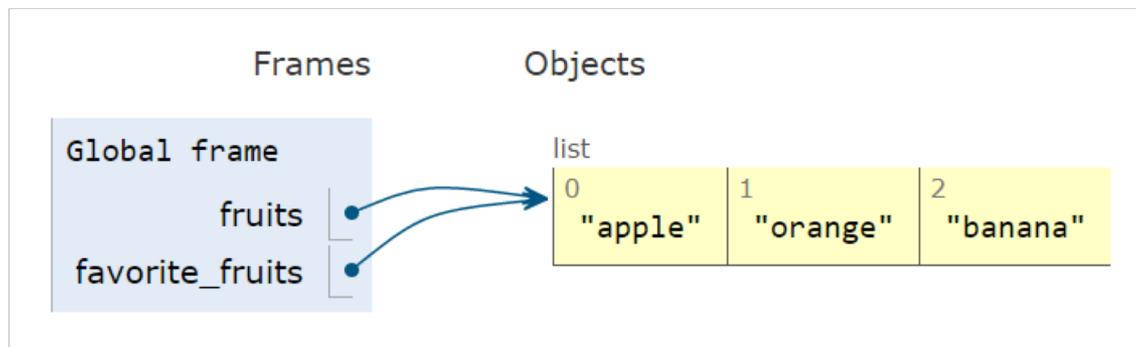
Copying Lists (For Real!)

```python
# option 1: slice
favorite_fruits = fruits[:]

#option 2: create a new list - best!
favorite_fruits = list(fruit)

#extend an empty list
favorite_fruits.extends(fruit)
```

Other operations with lists?   ⟶   Python docs

Printing a Dictionary

```python
legs = {"ant": 6, "snake": 0 }
print(legs)
```

```
{'ant': 6, 'snake': 0}
```

Modifying a Dictionary

```python
legs = {"ant": 6, "snake": 0 }
legs["spider"] = 273
```

```
{'ant': 6, 'snake': 0, 'spider': 273}
```

Modifying a Dictionary

```
legs = {"ant": 6, "snake": 0 }
legs["spider"] = 273 # basically, run!
legs["spider"] = 8 # better!
```

```
{'ant': 6, 'snake': 0, 'spider': 8}
```

Modifying a Dictionary

```
legs = {"ant": 6, "snake": 0, "spider": 8}

# I don't like spiders
del legs["spider"]

# Clear all the things!
legs.clear()
```

Retrieving a Value from a Dictionary

```python
legs = {"ant": 6, "snake": 0}

# get "ant"!
legs["ant"] # 6

# get "spider"
legs["spider"]
```

Retrieving a Value from a Dictionary

```python
legs = {"ant": 6, "snake": 0}

# get "ant"!
legs["ant"] # 6

# get "spider"
legs["spider"]
# KeyError: spider
```

Retrieving a Value from a Dictionary

```python
legs = {"ant": 6, "snake": 0}

# check if "spider" is in the dictionary
"spider" in legs # False

# get "spider" without throwing errors
legs.get("spider") # None

# get "spider" with a custom value
legs.get("spider", "Not present")
```

Functions

```python
def say_hello():
    print("Hello!")
```
definition

```python
say_hello()
```
call

Functions with Parameters

```python
def say_hello_to(name):
    print("Hello", name)

say_hello_to("AmI students")
```

Default Parameter Values

```python
def say_hello_to(name="AmI"):
    print("Hello", name)


say_hello_to() # Hello AmI


say_hello_to("students") # Hello students
```

Returning Values

```python
def build_greetings(name="AmI"):
    return "Hello" + name


greeting = build_greetings()
print(greeting) # Hello AmI
```

Returning Multiple Values

```python
def build_greetings(name="AmI"):
    return ("Hello", name)


(greeting, person) = build_greetings()
print(greeting + " to " + person)
# Hello to AmI
```

Documenting Functions

```python
def build_greetings(name="AmI"):
    '''Build a greeting in the format
Hello plus a given name'''
    return ("Hello", name)
```

docstring

Modules

- A way to logically organize the code

- They are files consisting of Python code

  - they can define (and implement) functions, variables, etc.

  - typically, the file containing a module is called in the same way

    - e.g., the *math* module resides in a file named *math.py*

Importing a Module

```
import math # import the math module

print(math.pi) # print 3.141592…
```

```
from math import pi # import pi, only!

print(pi) # print 3.141592…
```

```
from math import * # import all the names

print(pi)                          DO NOT USE
```

Command Line Parameters

```python
from sys import argv

script, first = argv

print("The script is called:", script)
print("The parameter is:", first)
```

unpacking

```
> python my_script.py one
The script is called: my_script.py
The parameter is: one
```

Reading Files

```
from sys import argv

filename = argv[1]          ⟵ open the file
txt = open(filename)


print("Here's your file %r:", % filename)
print(txt.read())
                    ⟵ show the file content
```

Writing Files

```python
from sys import argv

filename = argv[1]
# open in write mode and empty the file
target = open(filename, "w")


# write a string into the file
target.write("This is the new content")


target.close() # close the file
```

References and Links

- Python Documentation, http://docs.python.org/3
- The Python Tutorial, http://docs.python.org/3/tutorial/
- Online Python Tutor, http://pythontutor.com
- *«Think Python: How to think like a computer scientist»*, 2nd edition, Allen Downey, Green Tea Press, Needham, Massachusetts
- «*Dive into Python 3*», Mark Pilgrim
- «Learning Python» (5th edition), Mark Lutz, O'Reilly

Questions?

01QZP AMBIENT INTELLIGENCE

Luigi De Russis

luigi.derussis@polito.it
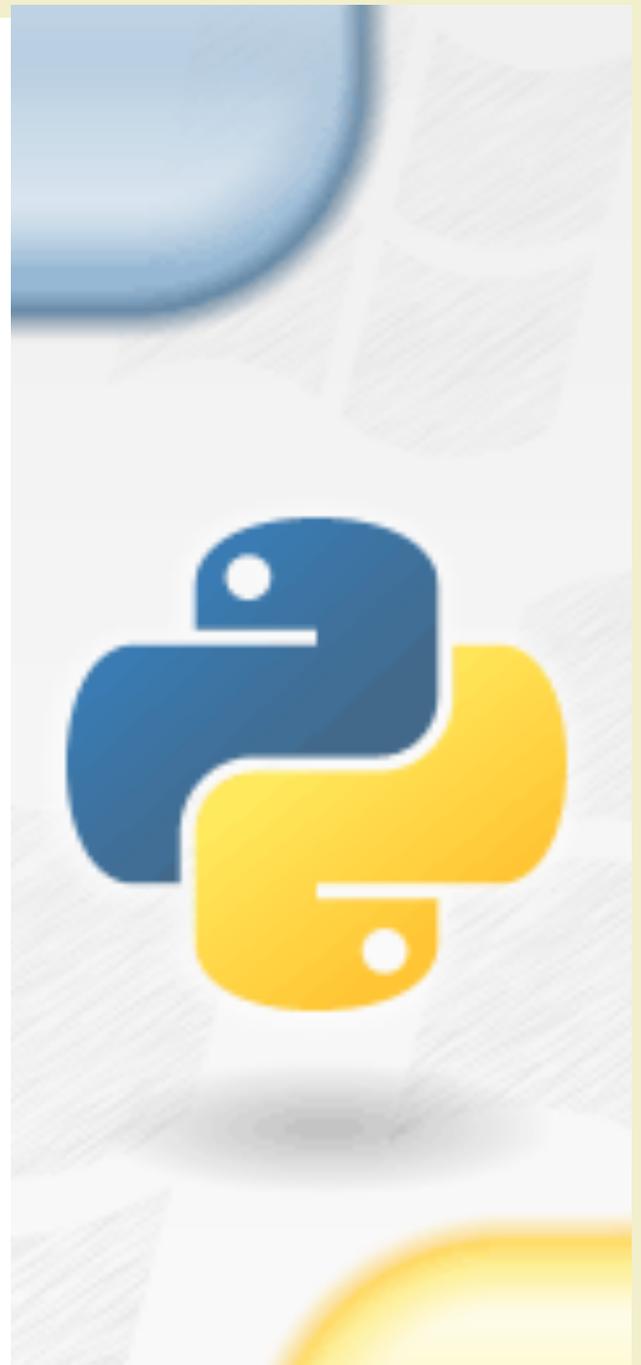
License

- This work is licensed under the Creative Commons "Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 4.0)" License.
- You are free:
  - to **Share** - to copy, distribute and transmit the work
  - to **Remix** - to adapt the work
- Under the following conditions:
  - **Attribution** - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  - **Noncommercial** - You may not use this work for commercial purposes.
  - **Share Alike** - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-sa/4.0/