Python3 Introduction

Gabriele Pozzati - gabriele.pozzati@scilifelab.se

— Objects, variables & operations

How to sum two numbers?

- Get one number	a = 0.0001012
- Get another number	b = 0.001012
- Sum them	a + b

Objects are data stored in memory.

Variables are “labels” referring to the assigned **objects**.
A to Z, a to z, 1 to 9, _, no start with number

Operations allow to create and manipulate **objects**.

— Data types

Data type	**Example**			
NoneType	None			
bool	True	False		
int	13	-147	89012932369127	
float	13.0	-147.7	8901293.2369127	1e5

— Data types

Data type	**Example**			
str	“do not the cat”	“1230601”	“True”	ITERABLES
list	[“do not”, “the cat”, 7, 1.35]			
tuple	(“do not”, “the cat”, 7, 1.35)			
dict	{“cat”:”Fuffi”, “dog”:”Toby”, “cow”:”Susette”}			

— Operators/Instructions

Arithmetic operator	**name**	**example**	**resulting object**
+	addition	8+3	11
-	subtraction	8-3	5
*	multiplication	8*3	24
/	division	8/3	2.6666666666666665
**	exponentiation	8**3	512
%	modulus	8%3	2
Comparison operator			
==	equal	8==3	False
!=	different	8!=3	True
>	major	8>3	True
<	minor	8<3	False
>=	major or equal	8>=3	True
<=	minor or equal	8<=3	False

— Operators/Instructions

Logical operator	example	resulting object
and	True and True	True
	True and False	False
	False and True	False
	False and False	False
or	True or True	True
	True or False	True
	False or True	True
	False or False	False
not	not False	True
	not True	False

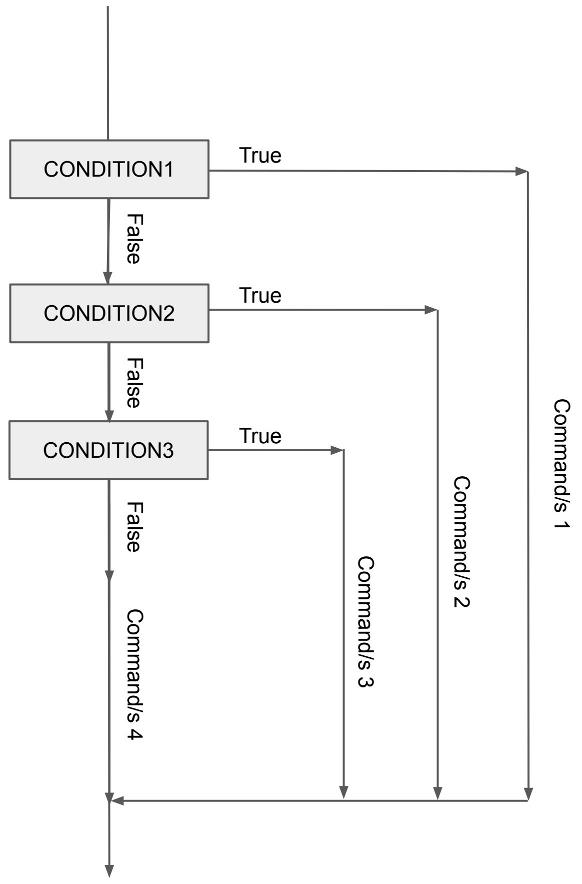
— Operators/Instructions

a = “do_not_the_cat”

Iterables indexing	**resulting object**
a[0]	“d”
a[4]	“o”
a[-1]	“t”
a[-3]	“c”
a[:3]	“do_”
a[3:]	“not_the_cat”
a[2:7]	“_not_”
a[-2:-6]	“”
a[1:6:1]	“o_not”
a[1:6:2]	“ont”
a[::-1]	“tac_eht_ton_od”
a[-2:-6:-1]	“ac_e”

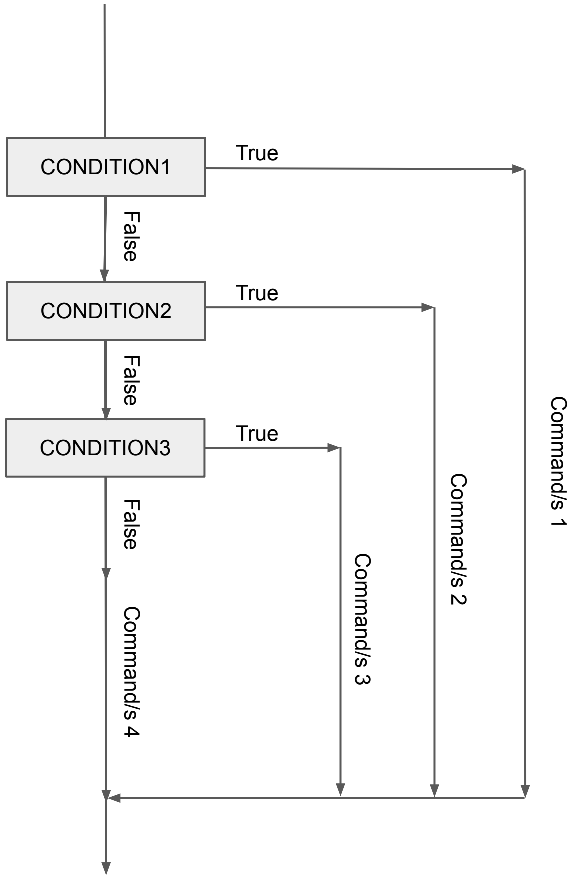
— Operators/Instructions

```
if      CONDITION1:
        execute command/s 1
        execute more command/s 1
elif    CONDITION2:
        execute command/s 2
        execute more command/s 2
elif    CONDITION3:
        execute command/s 3
        execute more command/s 3
…

else:
        execute command/s 4
        execute more command/s 4

commands to execute after if
```



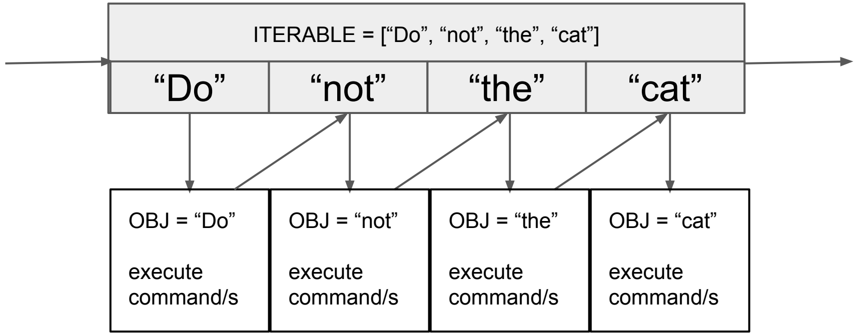
— Operators/Instructions

```
if      CONDITION1:
——————  execute command/s 1
——————  execute more command/s 1
elif    CONDITION2:
——————  execute command/s 2
——————  execute more command/s 2
elif    CONDITION3:
——————  execute command/s 3
——————  execute more command/s 3
…

else:
——————  execute command/s 4
——————  execute more command/s 4

commands to execute after if
```



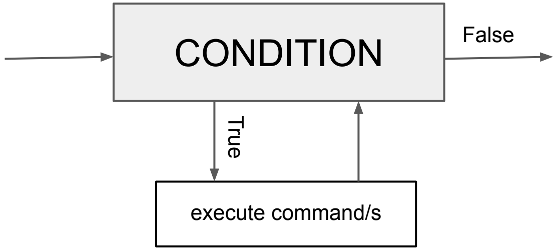
— Operators/Instructions

```
for OBJ in ITERABLE:
    execute command/s
    execute more command/s
commands to execute after iteration
```



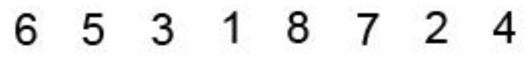
ITERABLE = ["Do", "not", "the", "cat"]
"Do"
"not"
"the"
"cat"
OBJ = "Do" execute command/s
OBJ = "not" execute command/s
OBJ = "the" execute command/s
OBJ = "cat" execute command/s

```
while CONDITION:
    execute command/s
    execute more command/s
commands to execute after iteration
```



— Algorithm design

Bubblesort



— Algorithm design

Bubblesort

```
a = [50,33,1,67,72,204,43,18]
```

```
if a[i-1] > a[i]:
    swap_variable = a[i-1]
    a[i-1] = a[i]
    a[i] = swap_variable
```

— Algorithm design

Bubblesort

```
a = [50,33,1,67,72,204,43,18]
idx = [1,2,3,4,5,6,7]
```

```
for i in idx:
    if a[i-1] > a[i]:
        swap_variable = a[i-1]
        a[i-1] = a[i]
        a[i] = swap_variable
```

— Algorithm design

Bubblesort

```
a = [50,33,1,67,72,204,43,18]
idx = [1,2,3,4,5,6,7]

swap_check = True
while swap_check:
    swap_check = False
    for i in idx:
        if a[i-1] > a[i]:
            swap_variable = a[i-1]
            a[i-1] = a[i]
            a[i] = swap_variable
            swap_check = True
```

— Algorithm design & functions

Bubblesort

```
def bubblesort(a, idx):
    swap_check = True
    while swap_check:
        swap_check = False
        for i in idx:
            if a[i-1] > a[i]:
                swap_variable = a[i-1]
                a[i-1] = a[i]
                a[i] = swap_variable
                swap_check = True
    return a

a = [50,33,1,67,72,204,43,18]
idx = [1,2,3,4,5,6,7]
a = bubblesort(a, idx)
```

— Algorithm design & functions

Bubblesort

```
def bubblesort(a):
    swap_check = True
    while swap_check:
        swap_check = False
        for i in range(1, len(a)):
            if a[i-1] > a[i]:
                swap_variable = a[i-1]
                a[i-1] = a[i]
                a[i] = swap_variable
                swap_check = True
    return a

a = [50,33,1,67,72,204,43,18]

a = bubblesort(a)
```

— Built-in funtions

Function	**Example**	**Resulting object**
print(OBJ)	print (“Hello world!”)	displays: Hello world!
type(OBJ)	type([1,2,3])	<class 'list'>
len(OBJ)	len(“Hello world!”)	12
range(int1, int2)	range(0,5)	0,1,2,3,4
str(), int(), list(), tuple()	str(-1985.7)	“-1985.7”
zip(iterable1, iterable2)	zip([4,8,2], [‘a’, ‘b’, ‘c’])	iterable <zip object at …>
enumerate(iterable)	enumerate([‘a’, ‘b’, ‘c’])	iterable <enumerate object at …>

— Classes

```
class Classname():
    def __init__(self, argument1, argument2, …):
        self.argument1 = argument1
        self.argument2 = argument2
        command1
        command2
        …

    def methodname(self, method_arg1, method_arg2):
        method command 1
        method command 2
        …
```

— Classes

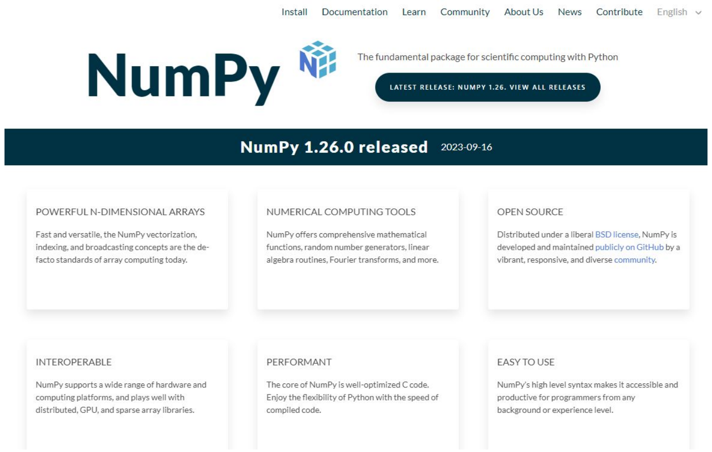
String methods

string.lower()
string.upper()
string.strip()
string.rstrip()
string.lstrip()
string.join()
string.split()

List Methods

list.append()
list.remove()
list.pop()
list.insert()
list.copy()
list.sort()

— Libraries



The fundamental package for scientific computing with Python

NumPy 1.26.0 released 2023-09-16

POWERFUL N-DIMENSIONAL ARRAYS

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

OPEN SOURCE

Distributed under a liberal BSD license, NumPy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

INTEROPERABLE

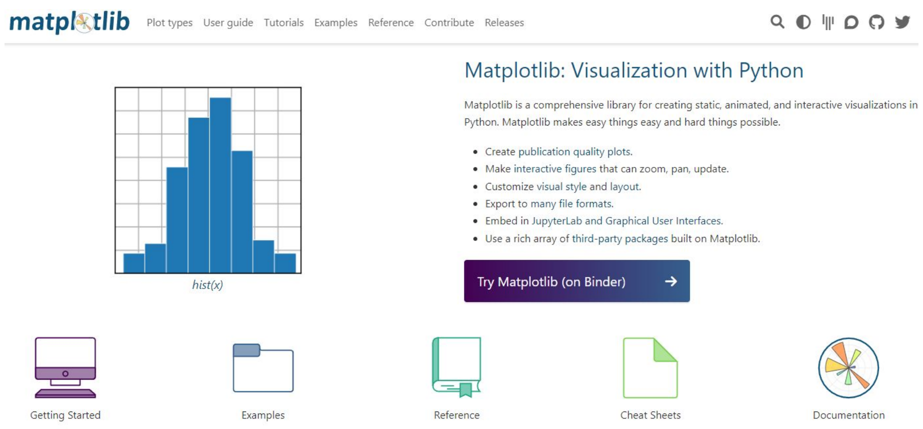
NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

PERFORMANT

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.



Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.