Python for datascience

track 1 - Python bases – 07th January 2022

Examen

Between 30m - 1h and 2h.

On last Saturday.

Program

Friday 07/01	4h	Python introduction + exercices
Friday 07/01	4h	Exercices + Pandas introduction
Saturday 08/01	4h	Pandas exercices
Friday 18/03	4h	Machine Learning introduction + exercices
Friday 18/03	4h	Pandas + ML exercices
Saturday 19/03	4h	Examen + exercices

Outline

1 Quick history

2 Python, how to

3 Bases

4 Hand’s on lab

0 Who am I?

Previously at kapten_.

Now a data engineering freelancer.

christophe.blefari@gmail.com

Data Engineer? Wut?

- Three data job exists as of today

    1. Data analysts

    2. Data scientists

    3. Data engineers

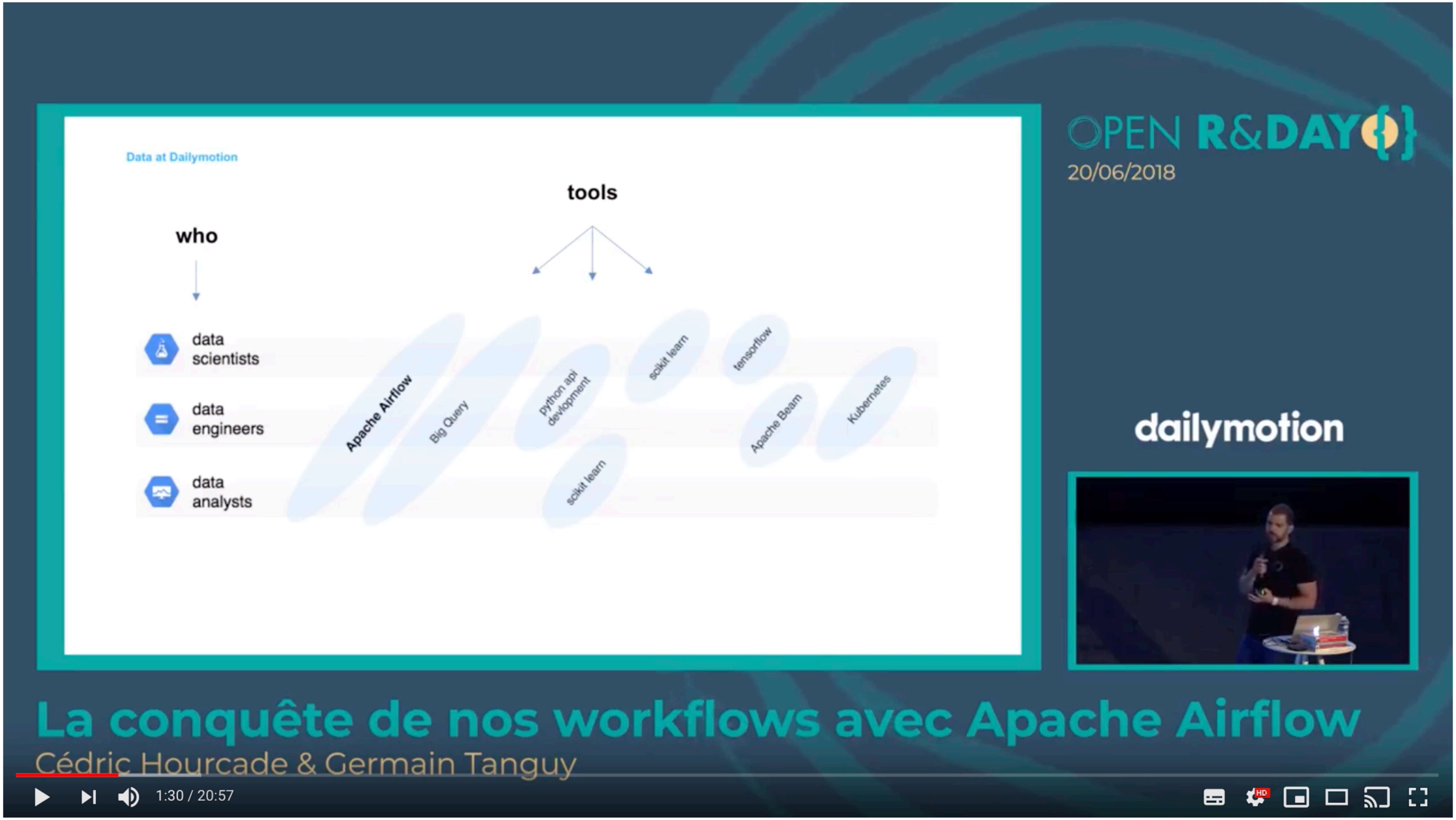
Data Engineer? Wut?

Data analysts work with data and answers to all business related questions like how many orders did we do last week? *etc.*

Data scientists are here to find correlation between data and develop models (machine learning based models for instance) that helps understand or predict the business.

Data engineers makes pipelines that get data from outside systems to the data systems.

Data Engineer? Wut?



Data at Dailymotion

tools

who

data scientists

data engineers

data analysts

Apache Airflow

Big Query

python api development

scikit learn

tensorflow

Apache Beam

Kubernetes

scikit learn

OPEN R&DAY{}

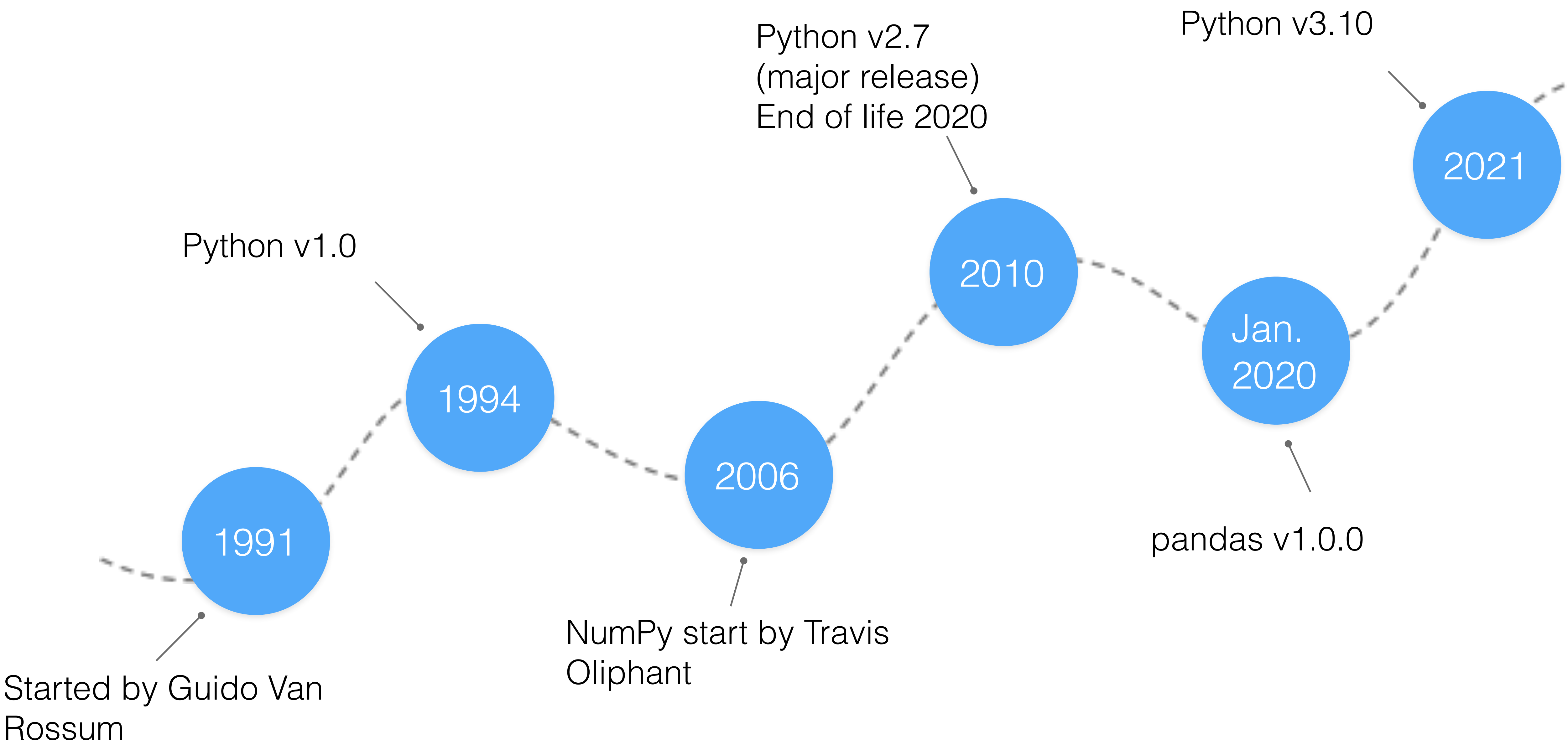
20/06/2018

dailymotion

La conquête de nos workflows avec Apache Airflow

Cédric Hourcade & Germain Tanguy

1 Quick history



1991

Started by Guido Van Rossum

1994

Python v1.0

2006

NumPy start by Travis Oliphant

2010

Python v2.7
(major release)
End of life 2020

Jan. 2020

pandas v1.0.0

2021

Python v3.10

1 Quick history: remarks

- There are a lot of web resources about Python
- Use conventions (PEP8, black - formatter)
- Versioning your code (git)
- You can start learning Python with 3.9 or 3.10.

1 Quick history: pros/cons

- Pros:
  - Big community
  - Simple and fast learning: productivity gain
  - Easy to script, to read
  - Lot of packages on PyPI for everything
  - Interpreted

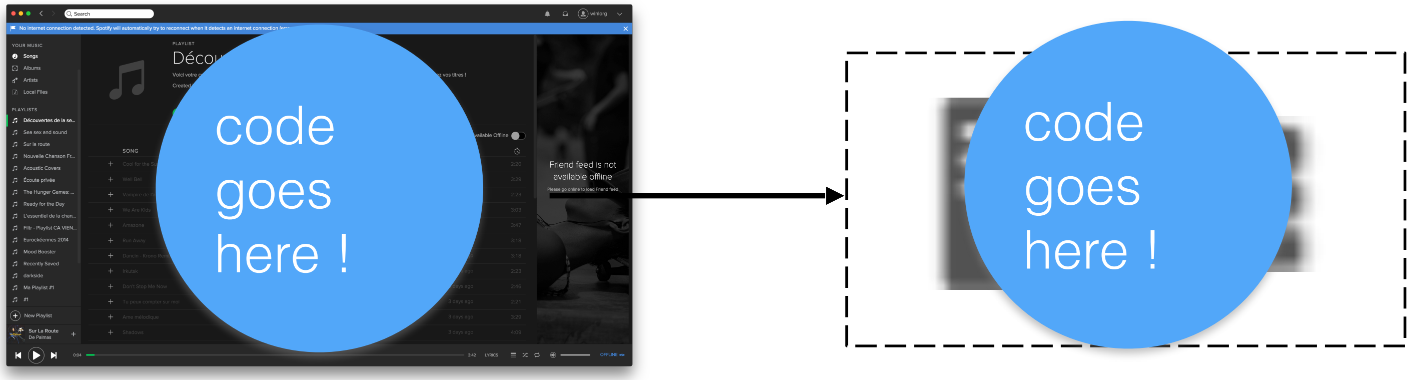
1 Quick history: pros/cons

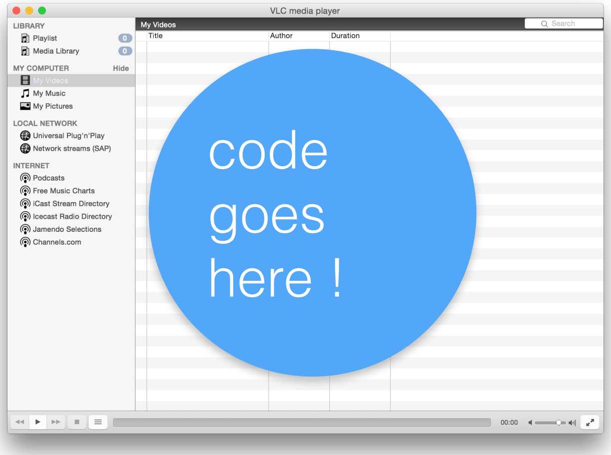
- Cons:
  - No real typing (: speed issues)
  - Performance (Python is high-level)

2 Global context

you often use **local** computer

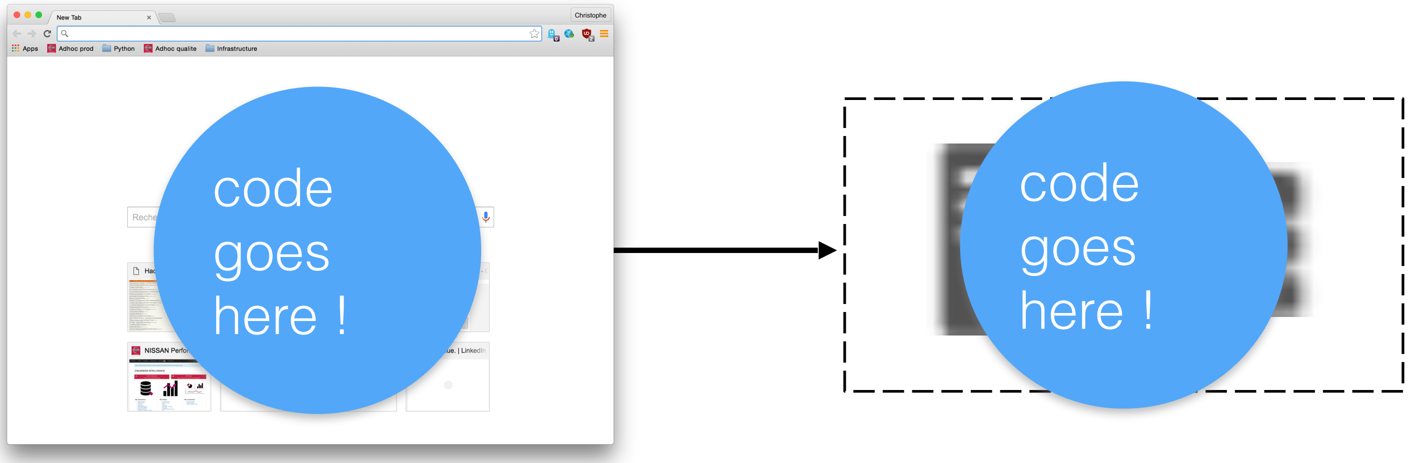
for your own activities





but it connects to **distant** servers

to make things happens
(ex: get data)



2 Global context

local machine

distant machines called servers

Every machine has an unique id that is the **MAC address**
And a network id that is the **IP address**

ssh, telnet, tcp →

we use **terminals** to execute programs
more generally to do everything

we write code with program called **IDE** or text editors

we **release** and **deploy** code on distant machines with versionning tools like Git, SVN and deployment tools like Ansible, Chief, etc. or custom scripts

Today will see **Notebooks** that are web interfaces where it's easy to develop, actually the code is executed on a machine but from the web browser with a socket mechanism between the browser and the machine

Distant machine are kept in **datacenters**.

Several providers of distant machines like:

- aws, google cloud engine, microsoft azure
- ovh, 1&1, digital ocean, gandhi
- jolicloud, heroku

On a distant machine you can install everything from scratch like on your computer:

- windows, linux in graphical or server mode

When an OS is installed you can start hacking with your favourite language.

On the OS you can install software like databases solutions (MySQL, PostgreSQL, Elastic Search, etc.).

Remember: a server is just a computer that is not on your desk

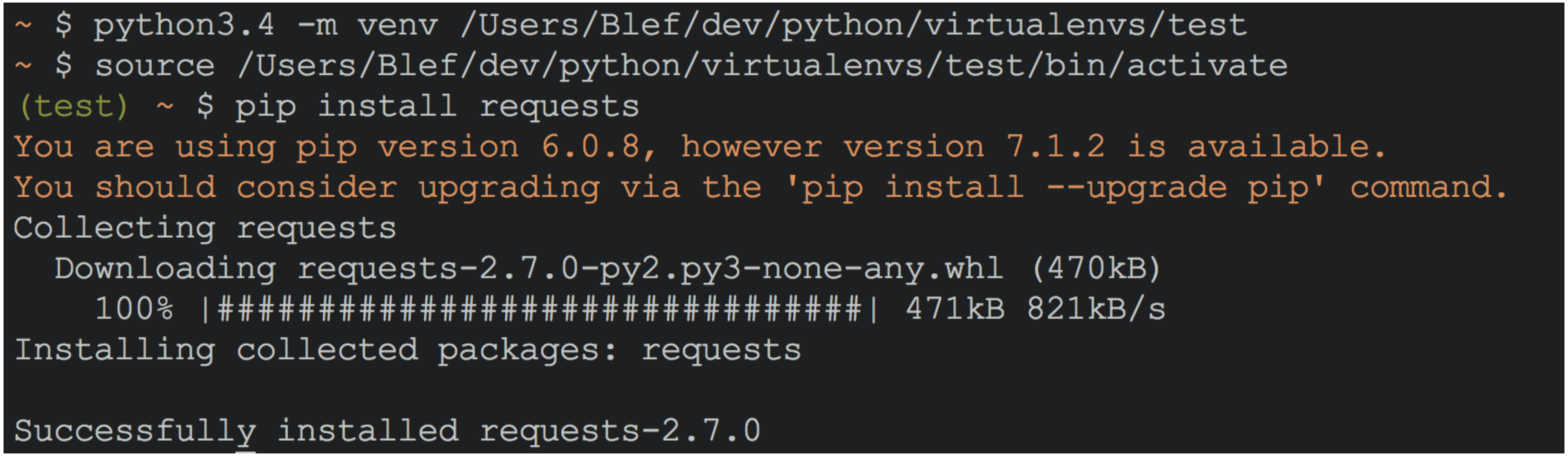
2 Python, how to use?

- You can use Python from
  - command line by typing **`python`**
  - ipython notebook server
  - Anaconda packages

- With virtualenv to isolate your packages versions

2 Python, how to develop

- You need to use **virtualenvs** to separate your different projects with different versions of packages



```
~ $ python3.4 -m venv /Users/Blef/dev/python/virtualenvs/test
~ $ source /Users/Blef/dev/python/virtualenvs/test/bin/activate
(test) ~ $ pip install requests
You are using pip version 6.0.8, however version 7.1.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting requests
  Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
    100% |################################| 471kB 821kB/s
Installing collected packages: requests

Successfully installed requests-2.7.0
```

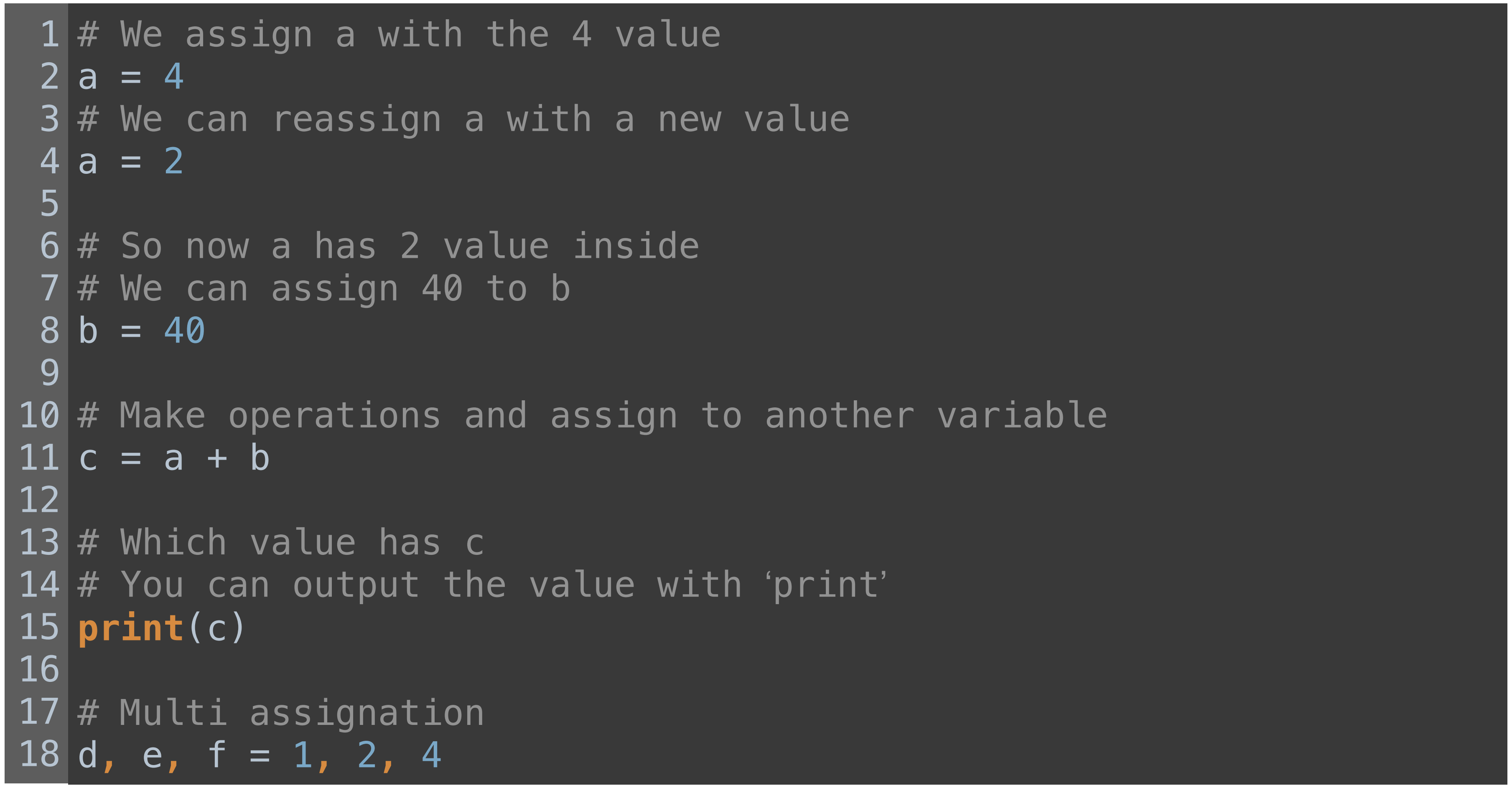
2 Python, how to develop

- We'll use iPython notebook because it's more easier to develop with for data analysis (and for one TD one IDE).
- But, for development we develop scripts inside files and execute them from command line:
- `$ python file.py`
- So, run your notebook server with anaconda
- Yo also can develop with programs likes
  - sublime-text, notepad(++), IDE, vim/emacs…

2 Python, how to debug

- Do debug! DO!
- Go on Google
- Read exceptions
- Don’t be afraid ; time and experience with programming will give you reflexes

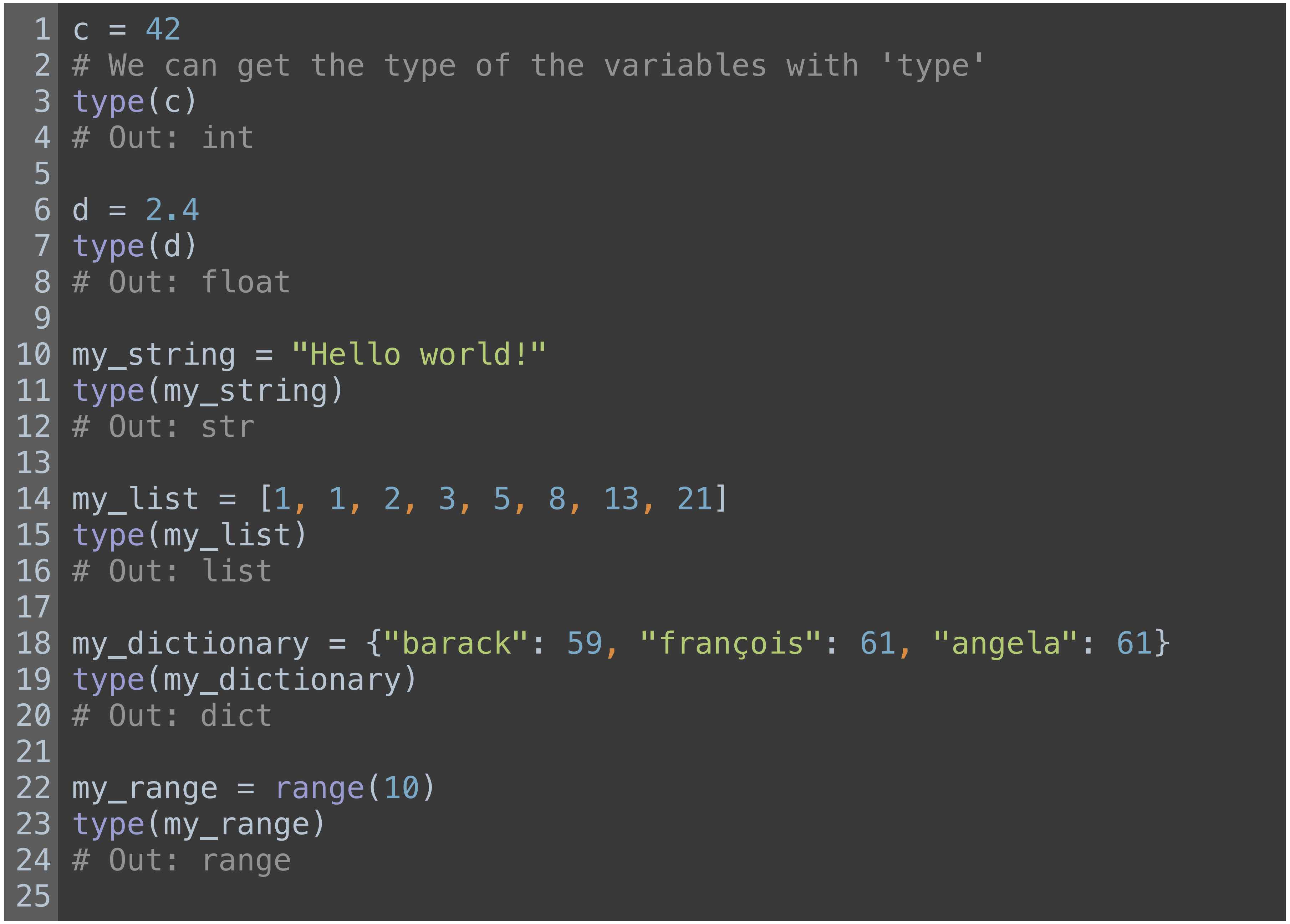
3 Bases: Variables



```python
# We assign a with the 4 value
a = 4
# We can reassign a with a new value
a = 2

# So now a has 2 value inside
# We can assign 40 to b
b = 40

# Make operations and assign to another variable
c = a + b

# Which value has c
# You can output the value with ‘print’
print(c)

# Multi assignation
d, e, f = 1, 2, 4
```

- variable name can’t start with a number
- By convention:
  - constants are in CAPS
  - names are > 3 letters
  - names that make sense
  - spaces around ‘=‘
  - space after comma ‘, toto’
  - use clear names

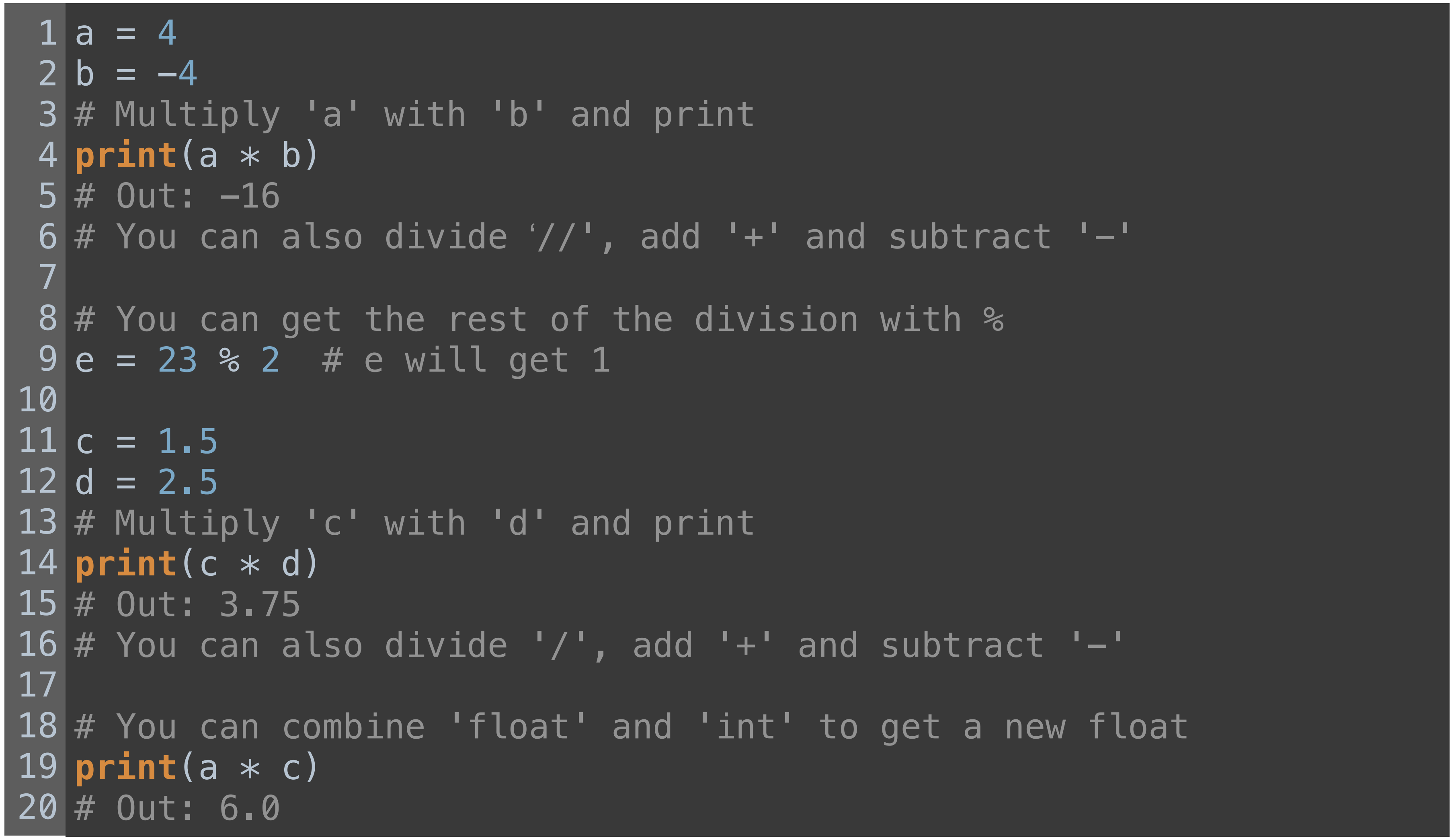
3 Bases: Types



```python
c = 42
# We can get the type of the variables with 'type'
type(c)
# Out: int

d = 2.4
type(d)
# Out: float

my_string = "Hello world!"
type(my_string)
# Out: str

my_list = [1, 1, 2, 3, 5, 8, 13, 21]
type(my_list)
# Out: list

my_dictionary = {"barack": 59, "françois": 61, "angela": 61}
type(my_dictionary)
# Out: dict

my_range = range(10)
type(my_range)
# Out: range

```

- Null element is **None** with a cap
- Python interpreter lets you make calculs

3 Bases: Types operations (int + float)

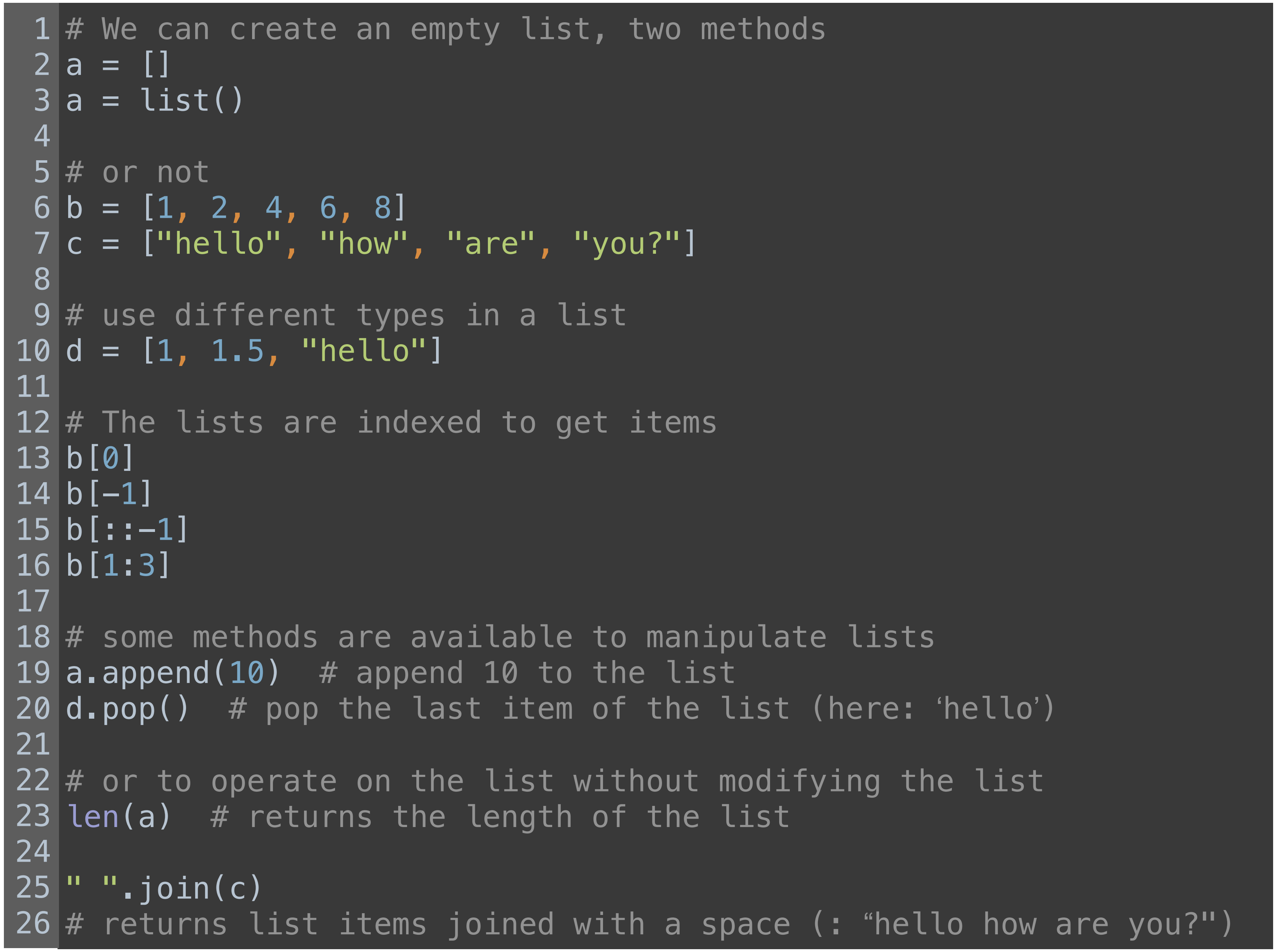


```python
a = 4
b = -4
# Multiply 'a' with 'b' and print
print(a * b)
# Out: -16
# You can also divide ‘//', add '+' and subtract '-'

# You can get the rest of the division with %
e = 23 % 2  # e will get 1

c = 1.5
d = 2.5
# Multiply 'c' with 'd' and print
print(c * d)
# Out: 3.75
# You can also divide '/', add '+' and subtract '-'

# You can combine 'float' and 'int' to get a new float
print(a * c)
# Out: 6.0
```

- Here we detailed base operations
- But you can combine variables to build a more complex expression

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

3 Bases: Types operations (list)

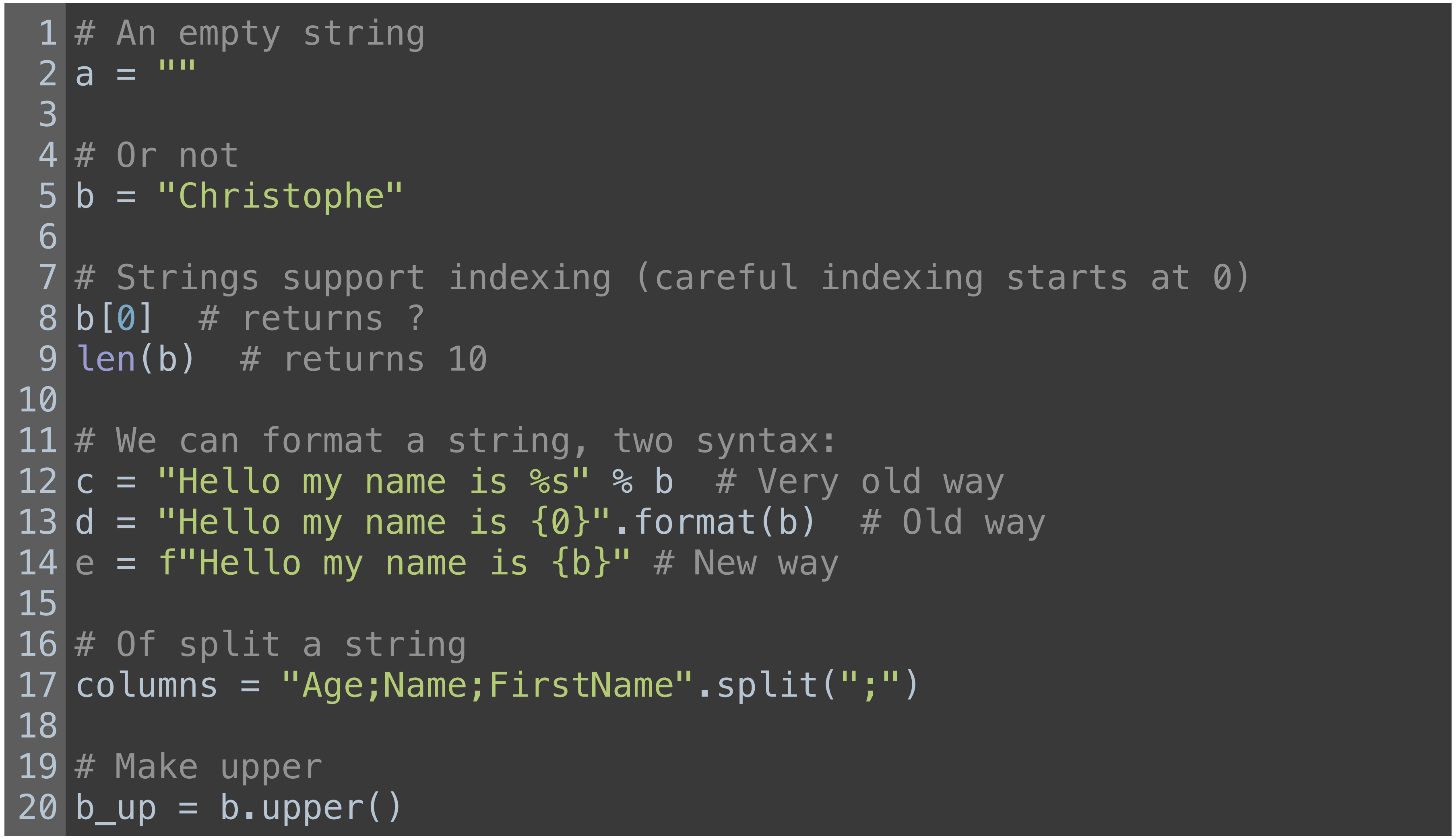


```
# We can create an empty list, two methods
a = []
a = list()

# or not
b = [1, 2, 4, 6, 8]
c = ["hello", "how", "are", "you?"]

# use different types in a list
d = [1, 1.5, "hello"]

# The lists are indexed to get items
b[0]
b[-1]
b[::-1]
b[1:3]

# some methods are available to manipulate lists
a.append(10)  # append 10 to the list
d.pop()  # pop the last item of the list (here: ‘hello’)

# or to operate on the list without modifying the list
len(a)  # returns the length of the list

" ".join(c)
# returns list items joined with a space (: “hello how are you?")
```

- Lists have lot of built-in methods
- We can **iterate** over a list (see after)
- We can **sort** a list
- Some methods:
  - Change (or not) the list
  - Return (or not) a result
- All types can be addable in a list (like a list for instance)
- A list is **always ordered**
- 🚧 Indexing starts at 0

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

3 Bases: Types operations (str)

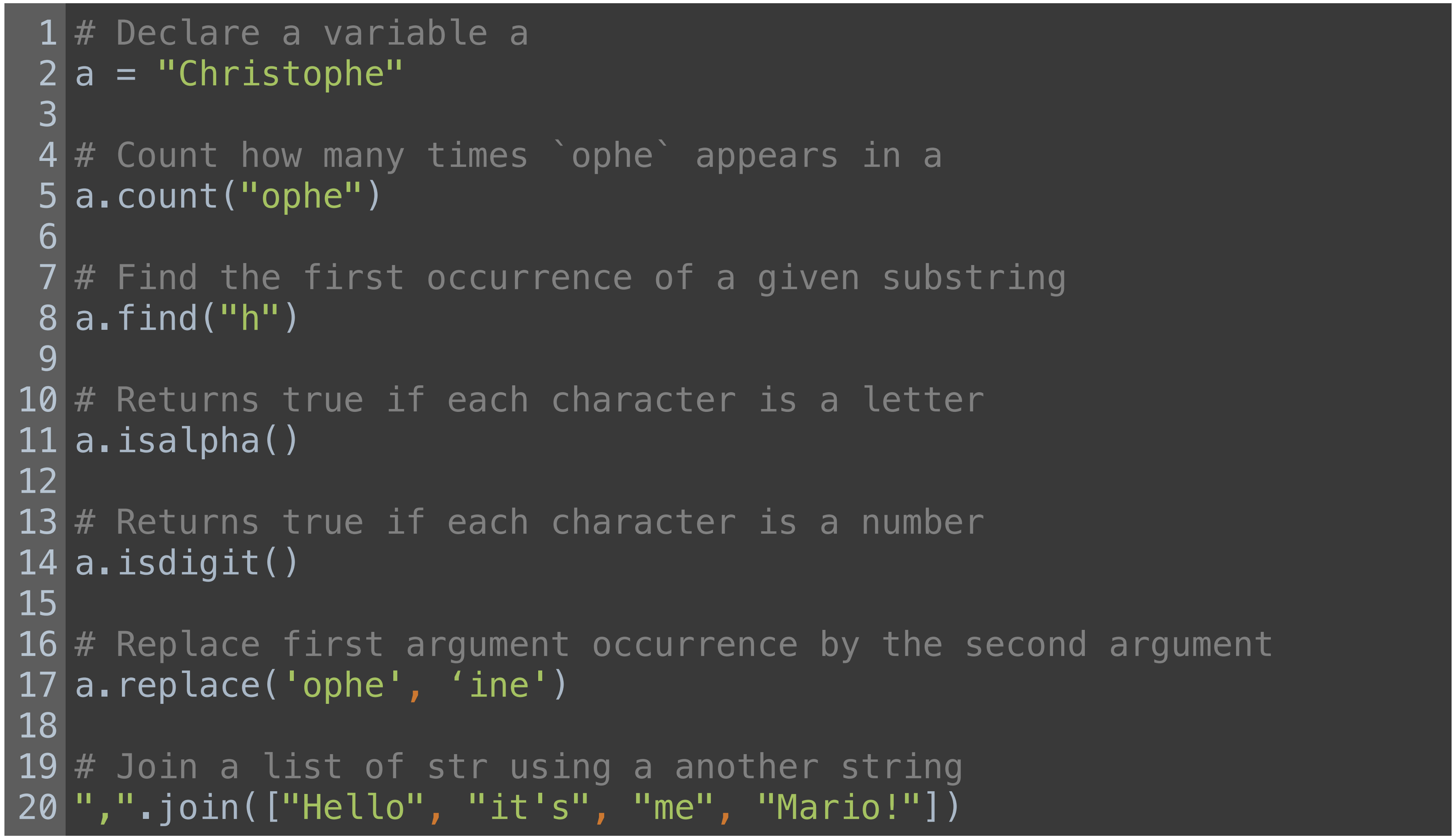


```python
# An empty string
a = ""

# Or not
b = "Christophe"

# Strings support indexing (careful indexing starts at 0)
b[0]  # returns ?
len(b)  # returns 10

# We can format a string, two syntax:
c = "Hello my name is %s" % b  # Very old way
d = "Hello my name is {0}".format(b)  # Old way
e = f"Hello my name is {b}" # New way

# Of split a string
columns = "Age;Name;FirstName".split(";")

# Make upper
b_up = b.upper()
```

- A str can be behaviour is like a list sometimes
  - Indexed (can get item and has a length)
  - Can be iterated
  - Can be sorted

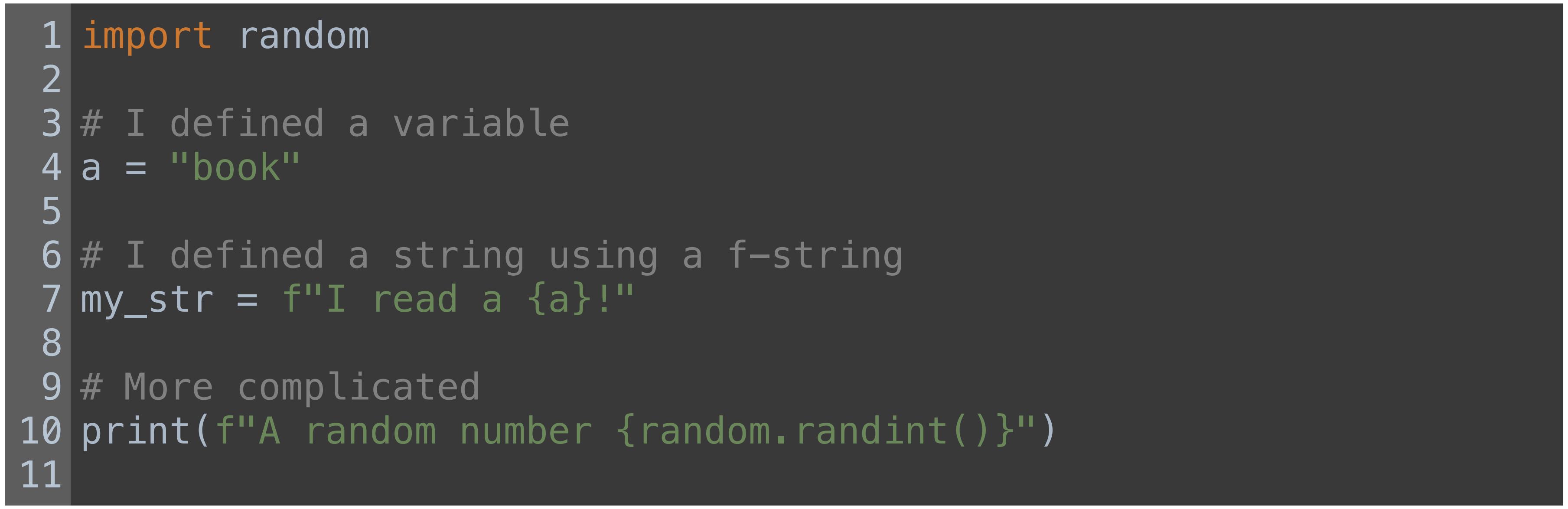
https://docs.python.org/3/library/stdtypes.html#textseq

3 Bases: Types operations (str)



```python
# Declare a variable a
a = "Christophe"

# Count how many times `ophe` appears in a
a.count("ophe")

# Find the first occurrence of a given substring
a.find("h")

# Returns true if each character is a letter
a.isalpha()

# Returns true if each character is a number
a.isdigit()

# Replace first argument occurrence by the second argument
a.replace('ophe', ‘ine')

# Join a list of str using a another string
",".join(["Hello", "it's", "me", "Mario!"])
```

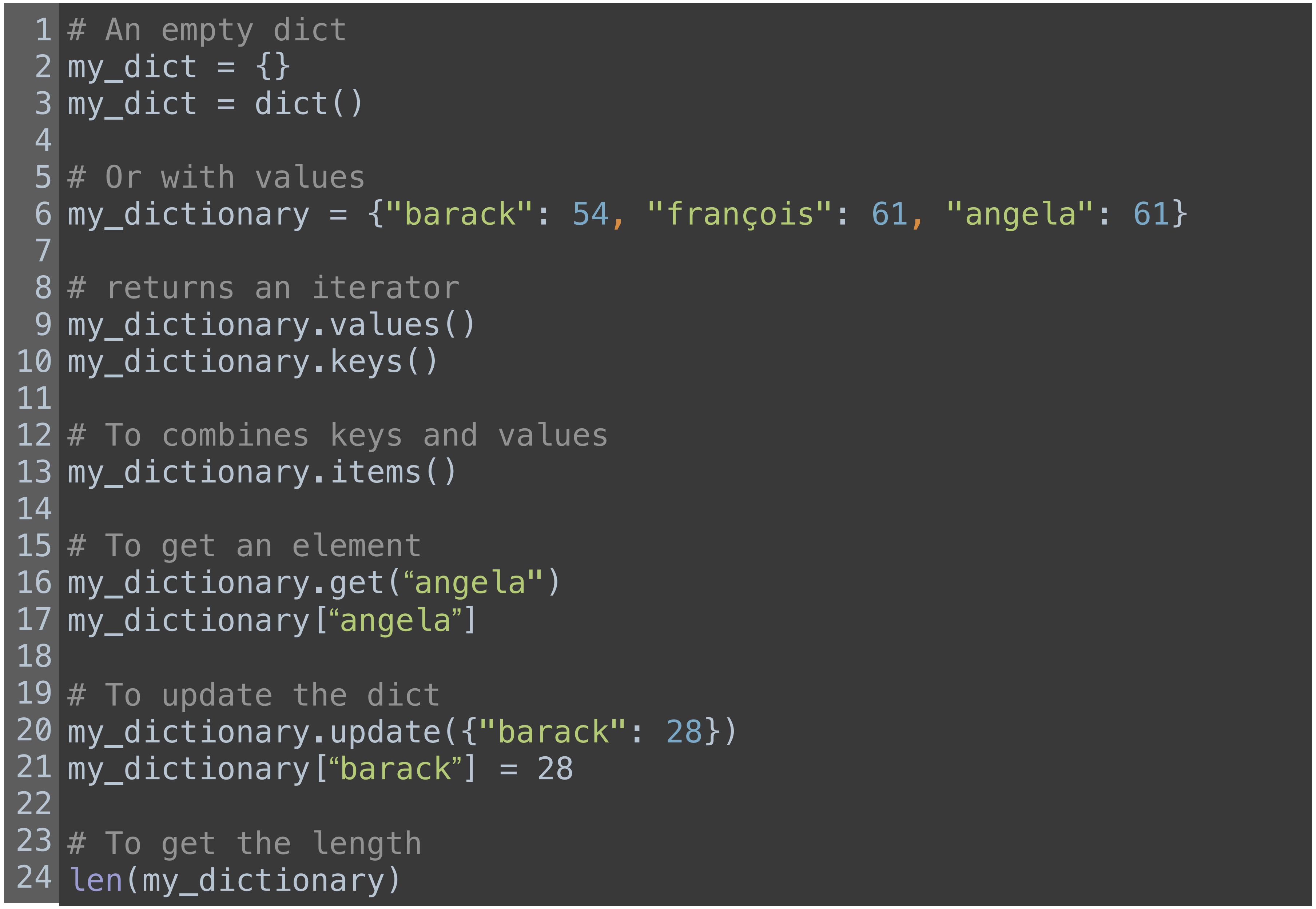
3 Bases: f-strings



```python
import random

# I defined a variable
a = "book"

# I defined a string using a f-string
my_str = f"I read a {a}!"

# More complicated
print(f"A random number {random.randint()}")

```

- fstrings are a new type of string that appears recently in Python
- fstrings are evaluated like a string but variable inside are interpreted

3 Bases: Types operations (dict)

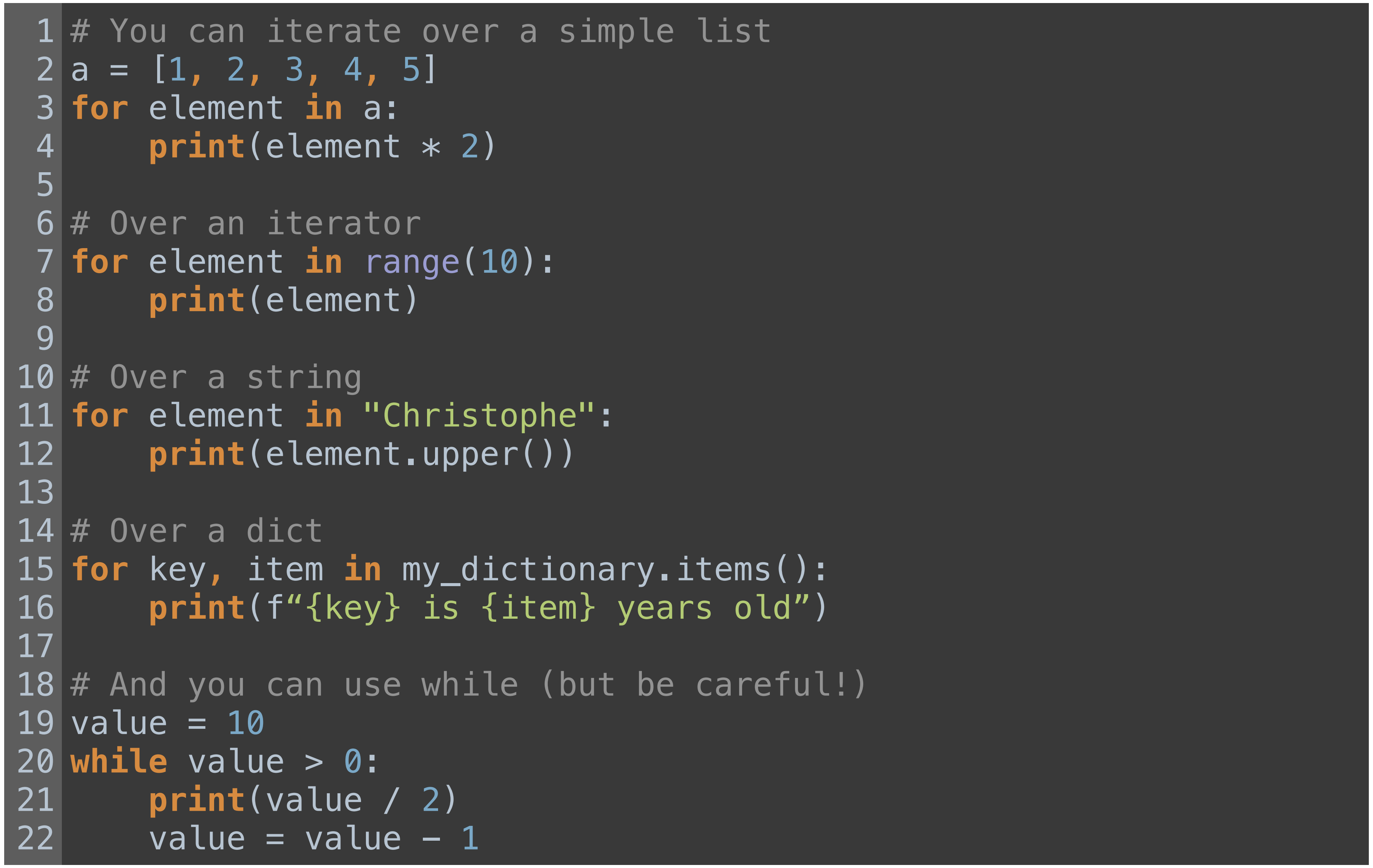


```python
# An empty dict
my_dict = {}
my_dict = dict()

# Or with values
my_dictionary = {"barack": 54, "françois": 61, "angela": 61}

# returns an iterator
my_dictionary.values()
my_dictionary.keys()

# To combines keys and values
my_dictionary.items()

# To get an element
my_dictionary.get(“angela")
my_dictionary[“angela”]

# To update the dict
my_dictionary.update({"barack": 28})
my_dictionary[“barack”] = 28

# To get the length
len(my_dictionary)
```

- You can do almost everything with Python base structures (list + dict)
- A dict **is not always ordered (depends on Python version)**

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

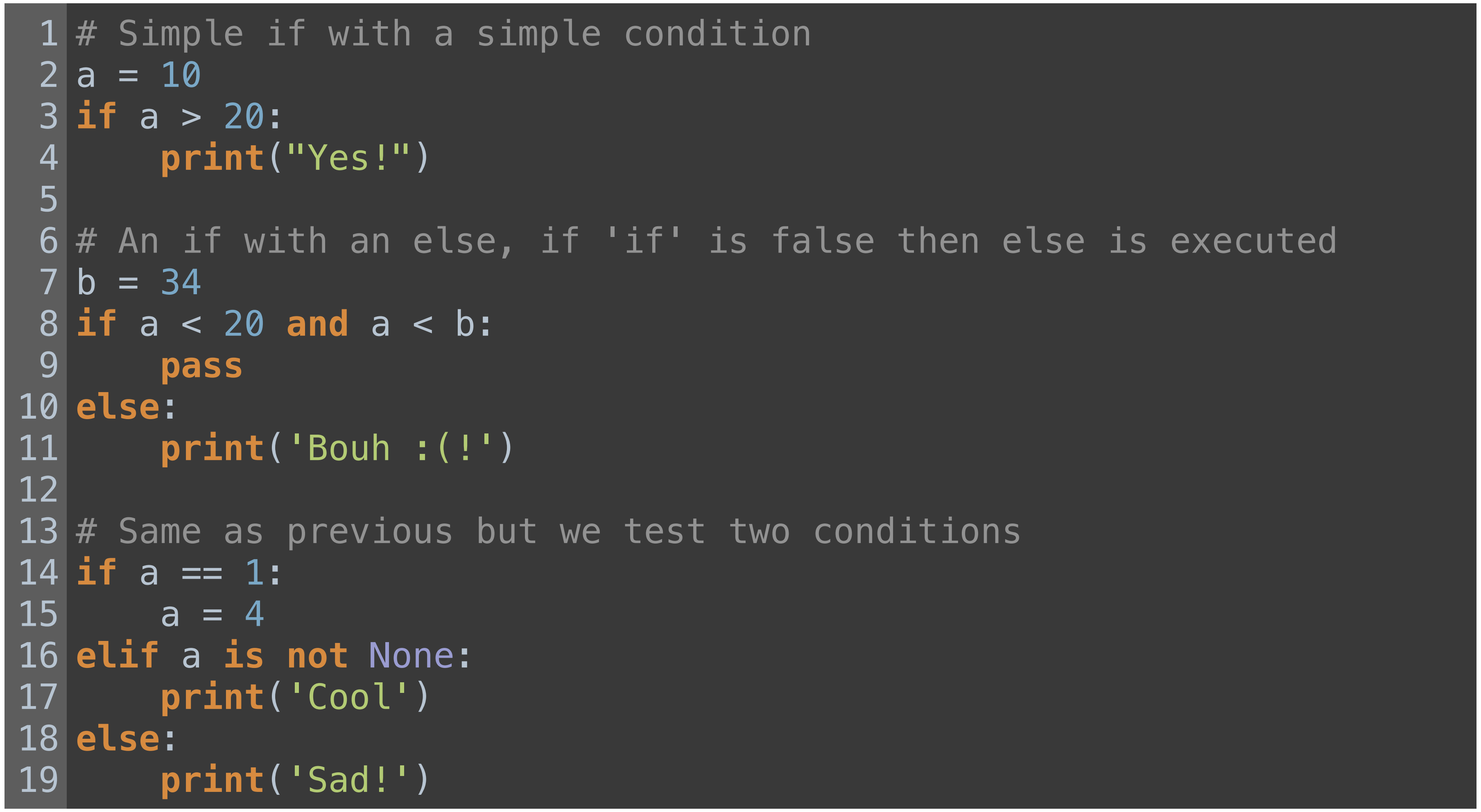
3 Bases: Loops



```python
# You can iterate over a simple list
a = [1, 2, 3, 4, 5]
for element in a:
    print(element * 2)

# Over an iterator
for element in range(10):
    print(element)

# Over a string
for element in "Christophe":
    print(element.upper())

# Over a dict
for key, item in my_dictionary.items():
    print(f“{key} is {item} years old”)

# And you can use while (but be careful!)
value = 10
while value > 0:
    print(value / 2)
    value = value - 1
```

- You can loop over lists or iterators
- element is a variable usable only in the ‘for’
- The indentation of the code is very important in Python

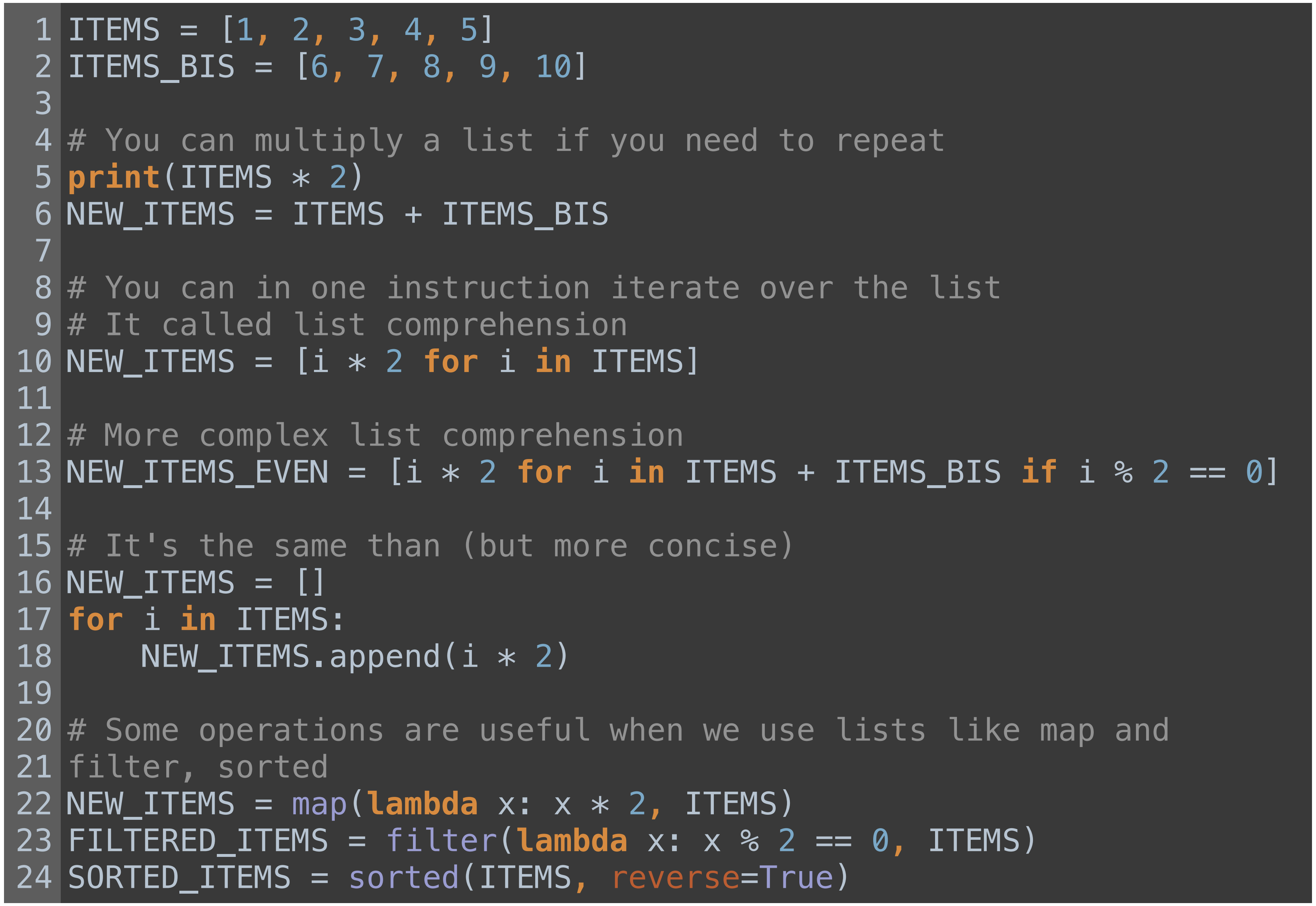
3 Bases: If statements



```python
# Simple if with a simple condition
a = 10
if a > 20:
    print("Yes!")

# An if with an else, if 'if' is false then else is executed
b = 34
if a < 20 and a < b:
    pass
else:
    print('Bouh :(!')

# Same as previous but we test two conditions
if a == 1:
    a = 4
elif a is not None:
    print('Cool')
else:
    print('Sad!')
```

- If statements are based on **boolean** value
- It’s only logic and you can combine everything to make **logical** expression:
    - or, and, in
    - not, is
    - >,<, <=, >=, ==
- Python syntax is verbose and simple
- You can use parenthesis to factorise expressions
- **T**rue and **F**alse are capitalised

3 Bases: Types operations (list) - advanced

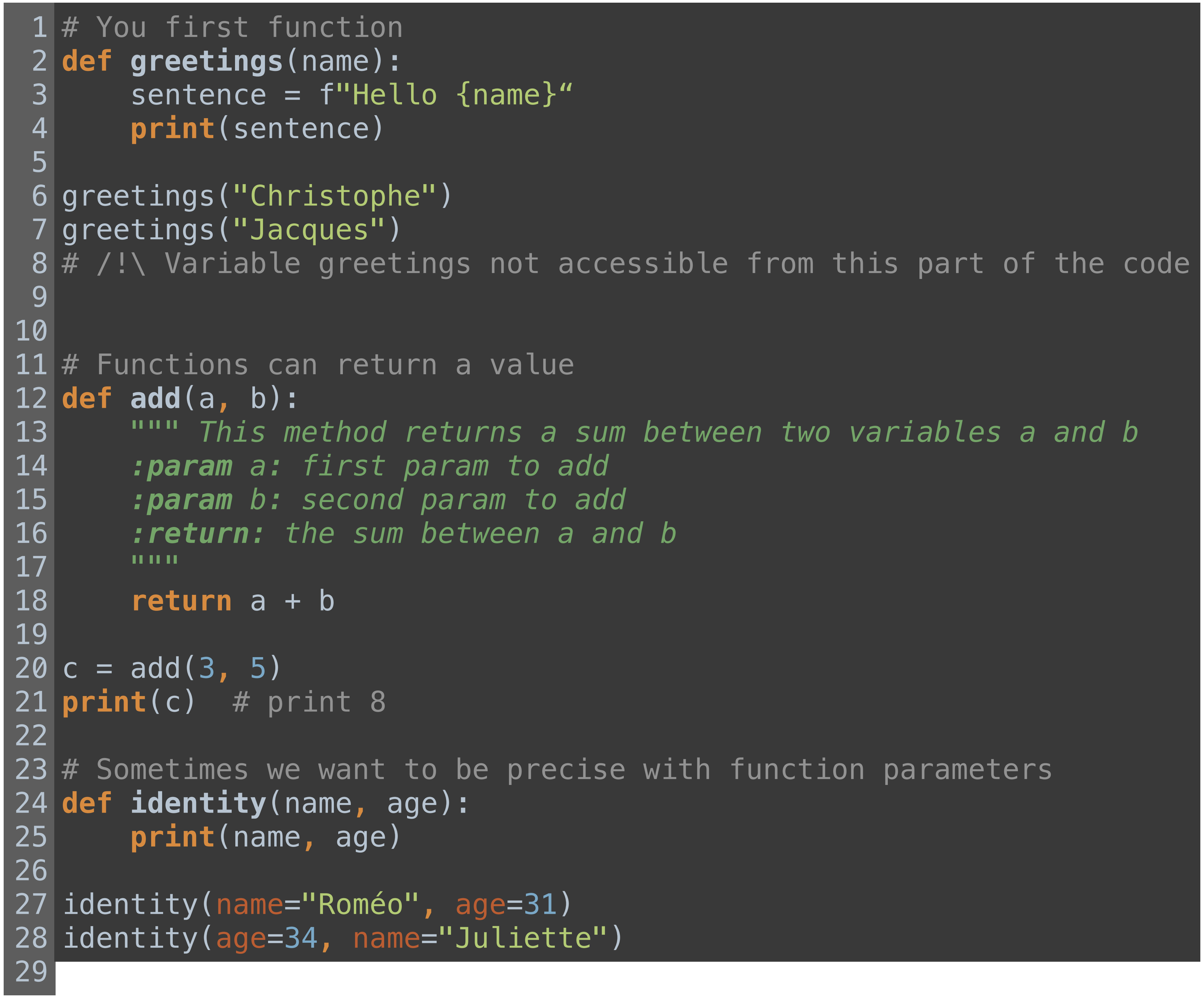


```python
ITEMS = [1, 2, 3, 4, 5]
ITEMS_BIS = [6, 7, 8, 9, 10]

# You can multiply a list if you need to repeat
print(ITEMS * 2)
NEW_ITEMS = ITEMS + ITEMS_BIS

# You can in one instruction iterate over the list
# It called list comprehension
NEW_ITEMS = [i * 2 for i in ITEMS]

# More complex list comprehension
NEW_ITEMS_EVEN = [i * 2 for i in ITEMS + ITEMS_BIS if i % 2 == 0]

# It's the same than (but more concise)
NEW_ITEMS = []
for i in ITEMS:
    NEW_ITEMS.append(i * 2)

# Some operations are useful when we use lists like map and
filter, sorted
NEW_ITEMS = map(lambda x: x * 2, ITEMS)
FILTERED_ITEMS = filter(lambda x: x % 2 == 0, ITEMS)
SORTED_ITEMS = sorted(ITEMS, reverse=True)
```

- Lists are the most useful structure in Python for data analyses
- You need to master the list operations
- Index starts at 0 (reminder!)

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

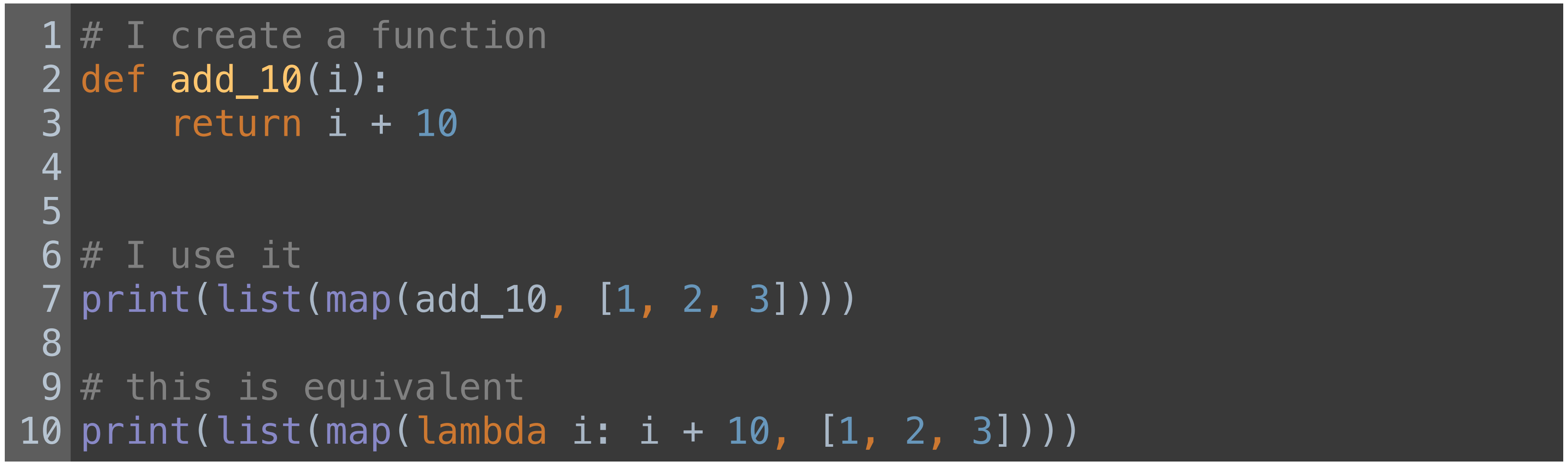
3 Bases: Functions



```python
# You first function
def greetings(name):
    sentence = f"Hello {name}“
    print(sentence)

greetings("Christophe")
greetings("Jacques")
# /!\ Variable greetings not accessible from this part of the code


# Functions can return a value
def add(a, b):
    """ This method returns a sum between two variables a and b
    :param a: first param to add
    :param b: second param to add
    :return: the sum between a and b
    """
    return a + b

c = add(3, 5)
print(c)  # print 8

# Sometimes we want to be precise with function parameters
def identity(name, age):
    print(name, age)

identity(name="Roméo", age=31)
identity(age=34, name="Juliette")
```

- Functions can return a value (if not it returns **None**)
- We can say also a **method** or a **procedure**
- By convention we give explicit names to functions and we comment a lot the code
- A function has a **signature**
    - Here add signature is to have ***a*** and ***b*** as **parameters**

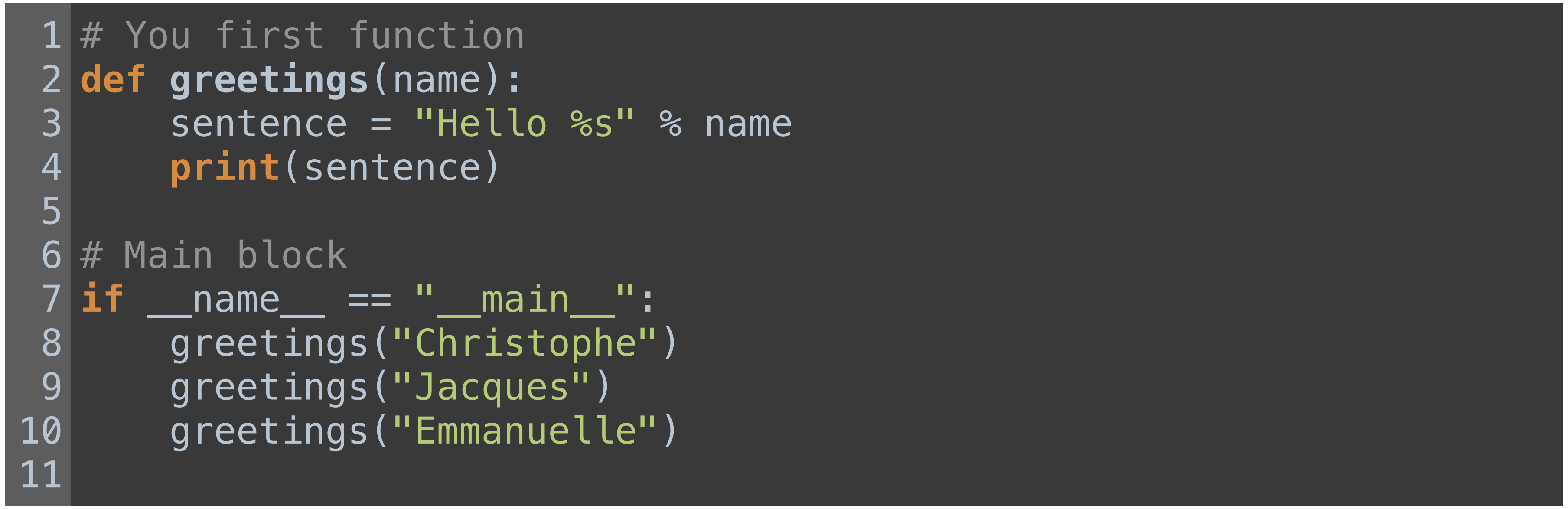
3 Bases: Lambda functions



```python
# I create a function
def add_10(i):
    return i + 10


# I use it
print(list(map(add_10, [1, 2, 3])))

# this is equivalent
print(list(map(lambda i: i + 10, [1, 2, 3])))
```

- A lambda is an anonymous function (meaning function without a name) and can be used sometimes in Python to simplify the code

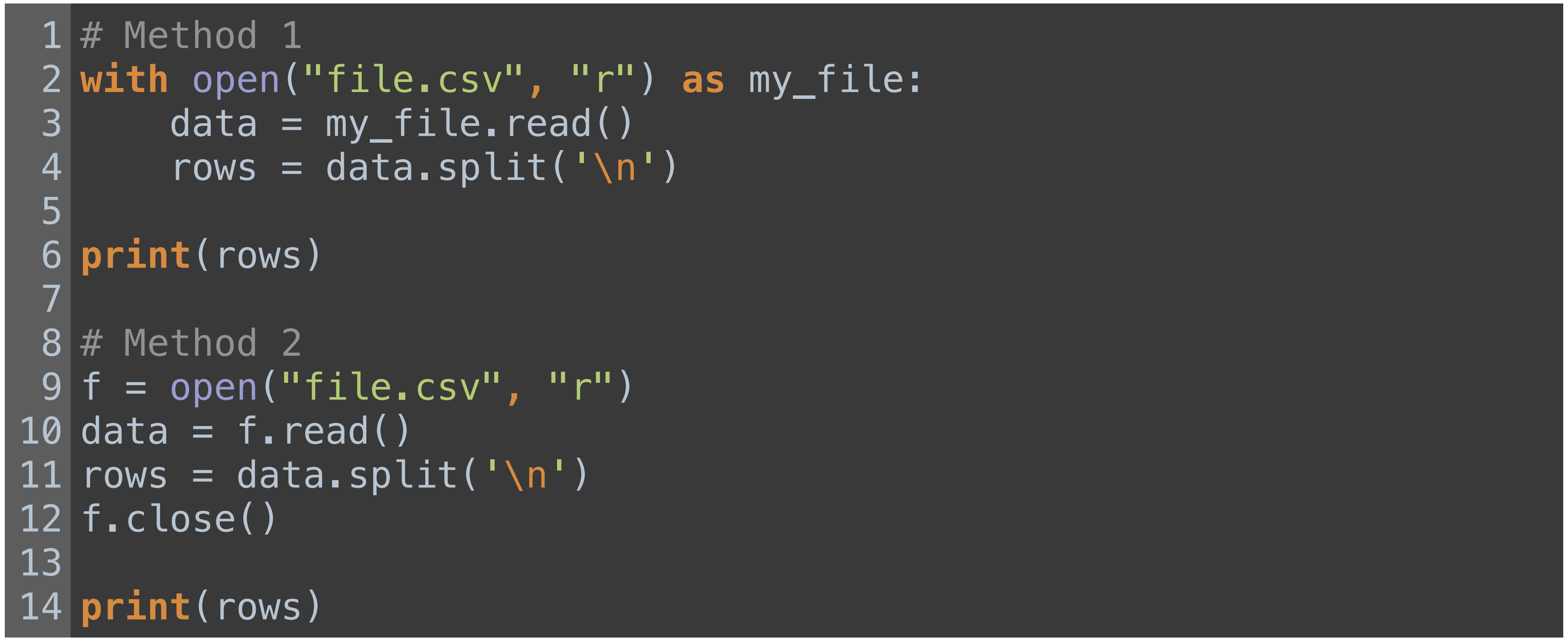
3 Bases: Main block



```python
# You first function
def greetings(name):
    sentence = "Hello %s" % name
    print(sentence)

# Main block
if __name__ == "__main__":
    greetings("Christophe")
    greetings("Jacques")
    greetings("Emmanuelle")

```

- We use the “main” block to separate the code execution and the import call
- i.e. if we import a module all code except the main will be executed
- If we call the file himself the main will be executed

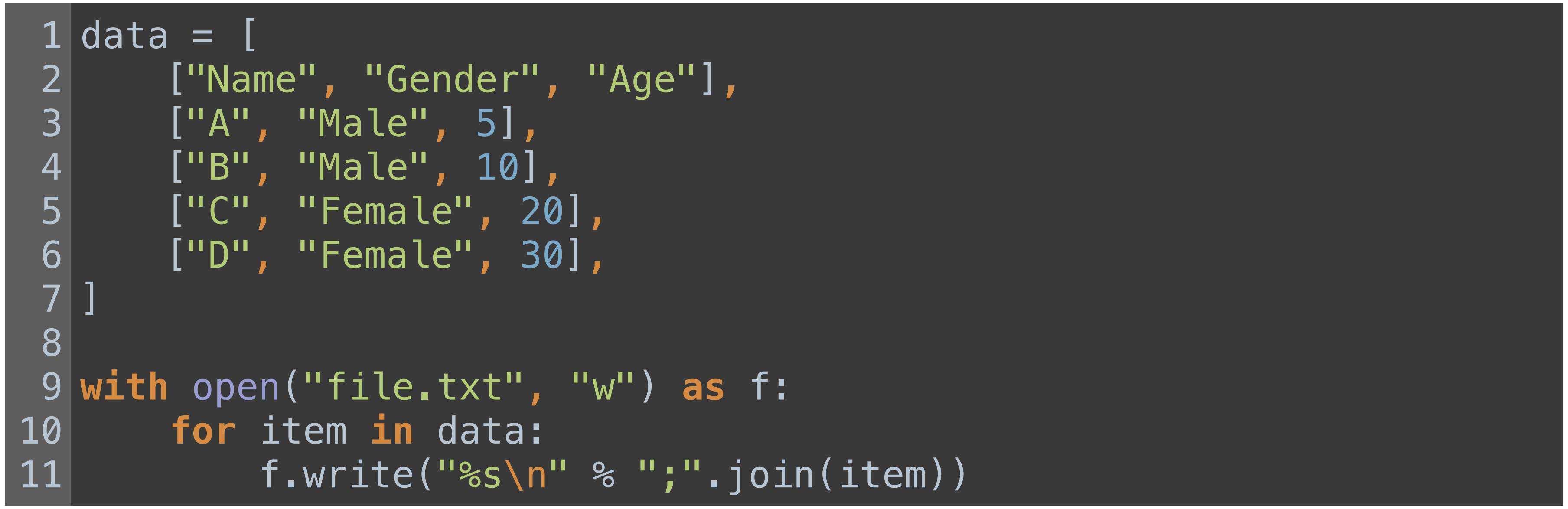
3 Bases: Open and read files



```python
# Method 1
with open("file.csv", "r") as my_file:
    data = my_file.read()
    rows = data.split('\n')

print(rows)

# Method 2
f = open("file.csv", "r")
data = f.read()
rows = data.split('\n')
f.close()

print(rows)
```

- There are two methods to **open** files, I prefer the first one presented here
- In data analyses you often use files, so this snippet on code is very important
- Next week we’ll see how to connect to other kinds of data (like databases or API)
- open() second parameter is the open mode: here we read the file so “r”

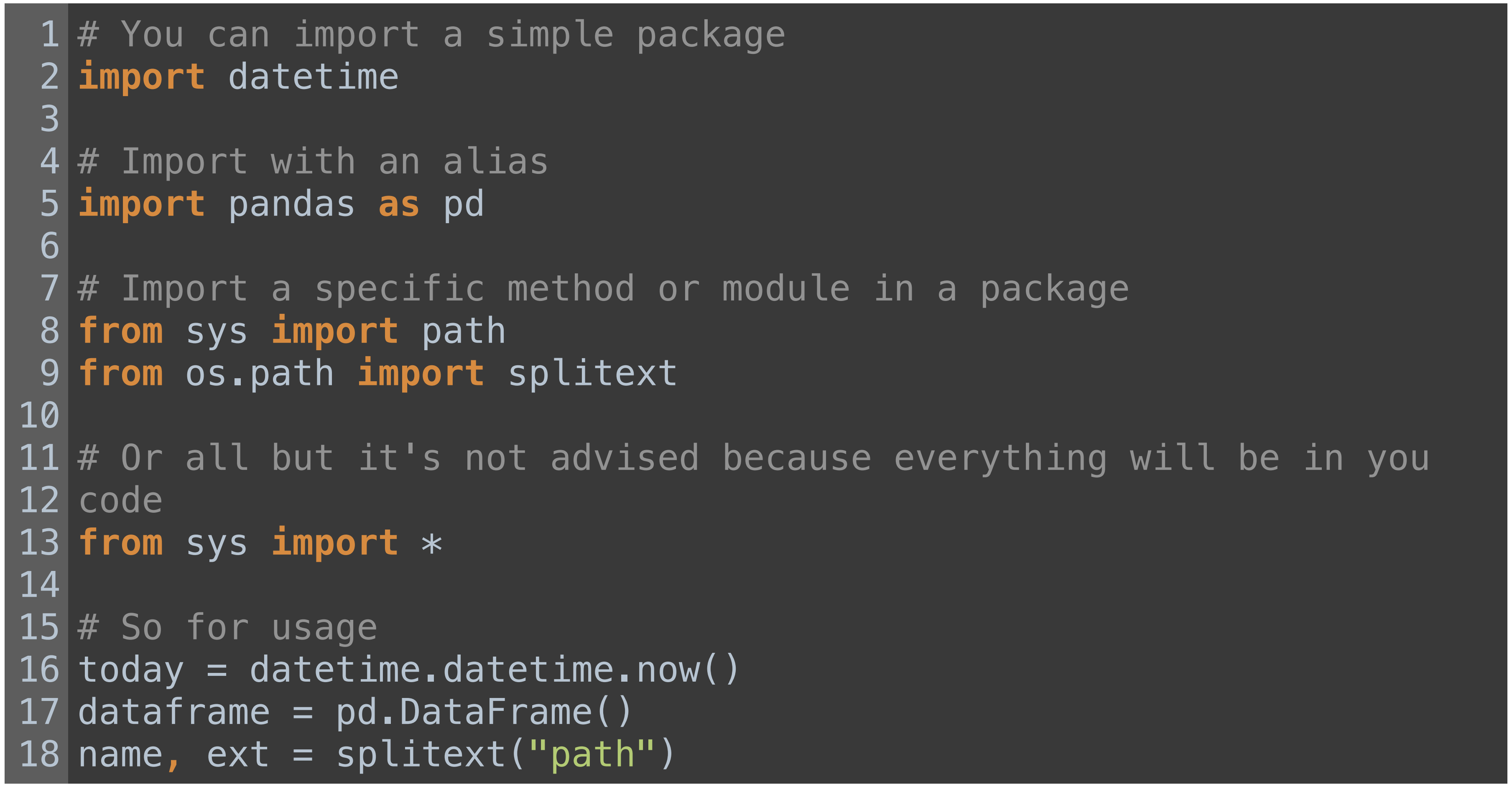
3 Bases: Open and write files



```python
data = [
    ["Name", "Gender", "Age"],
    ["A", "Male", 5],
    ["B", "Male", 10],
    ["C", "Female", 20],
    ["D", "Female", 30],
]

with open("file.txt", "w") as f:
    for item in data:
        f.write("%s\n" % ";".join(item))
```

- The file will be created in the path given
- Here we write the file so “w” for the open mode

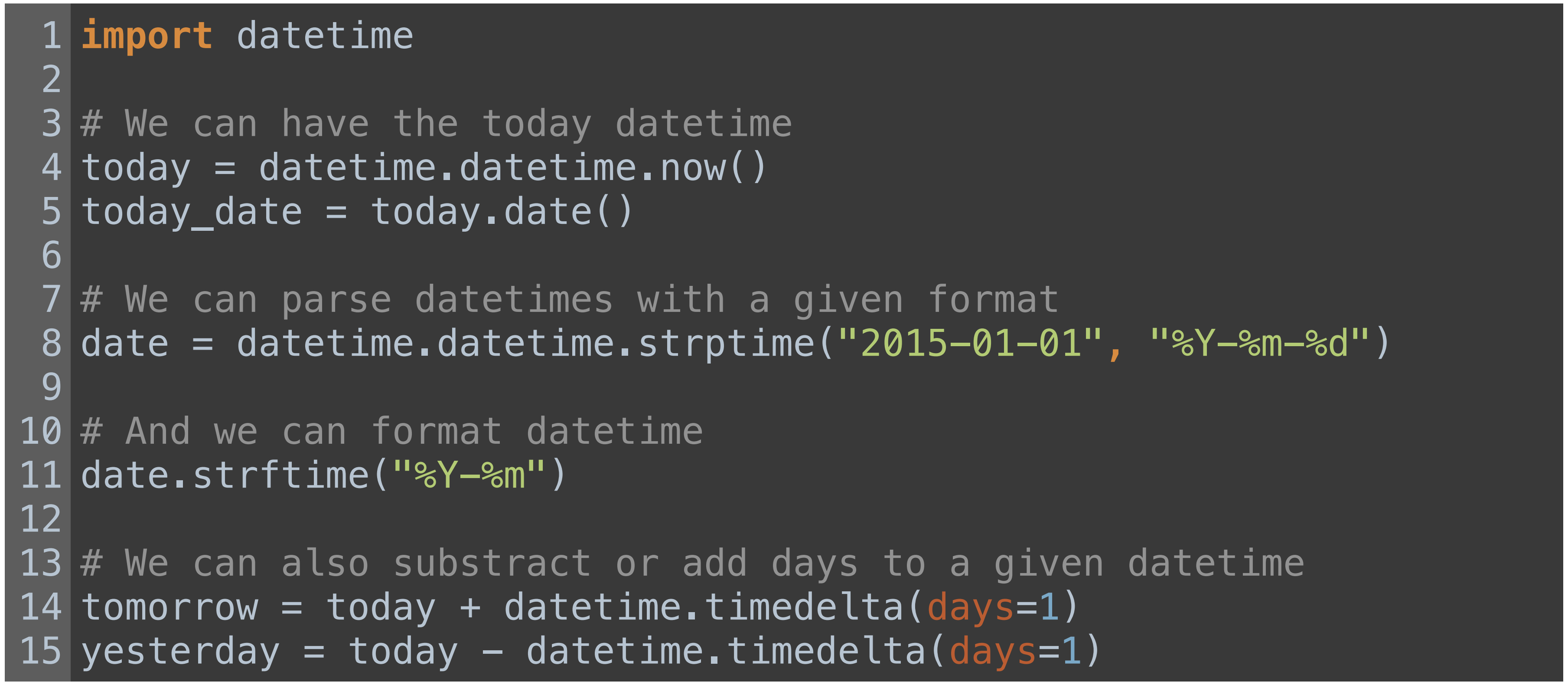
3 Bases: Imports



```python
# You can import a simple package
import datetime

# Import with an alias
import pandas as pd

# Import a specific method or module in a package
from sys import path
from os.path import splitext

# Or all but it's not advised because everything will be in you
code
from sys import *

# So for usage
today = datetime.datetime.now()
dataframe = pd.DataFrame()
name, ext = splitext("path")
```

- We you want to develop you will always have to use external packages **imports** are the key
- Hint: order your imports at the top of the file alphabetically

3 Bases: datetime



```python
import datetime

# We can have the today datetime
today = datetime.datetime.now()
today_date = today.date()

# We can parse datetimes with a given format
date = datetime.datetime.strptime("2015-01-01", "%Y-%m-%d")

# And we can format datetime
date.strftime("%Y-%m")

# We can also substract or add days to a given datetime
tomorrow = today + datetime.timedelta(days=1)
yesterday = today - datetime.timedelta(days=1)
```

- Date times formats are described in the official docs

3 Bases: Exceptions

ZeroDivisionError

```
>>> 4 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

TypeError

```
>>> 5 + "rrr"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

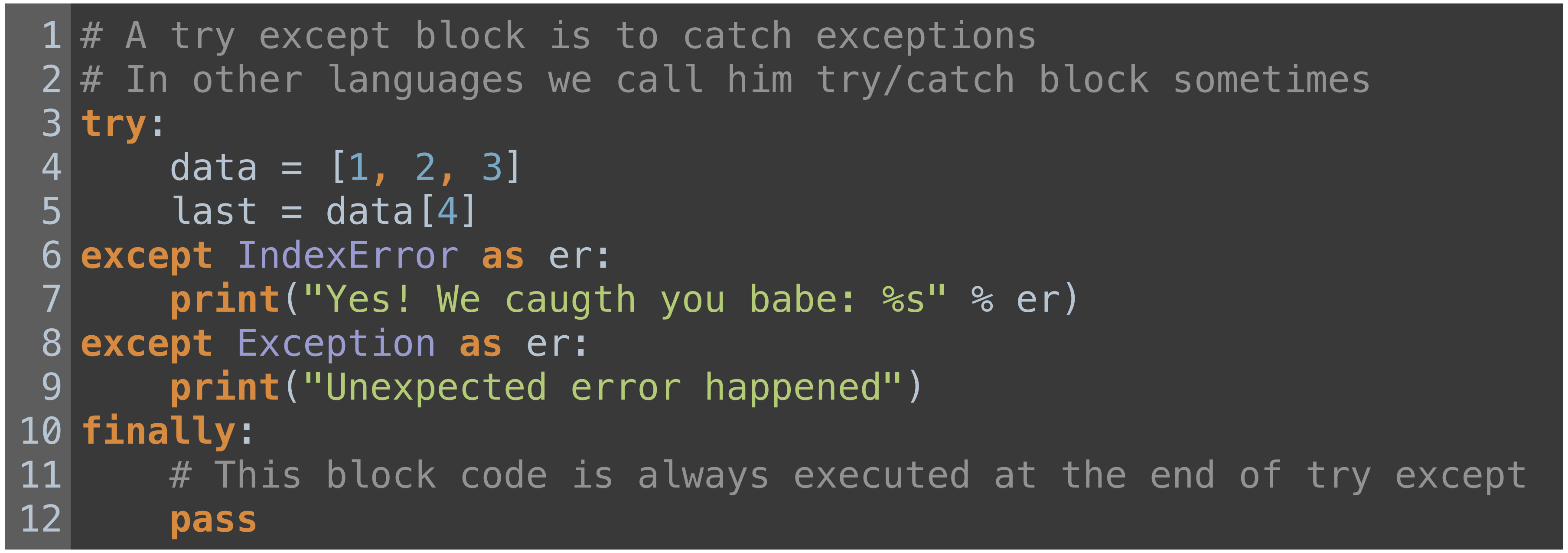
SyntaxError

```
>>> def hello(p1 p2):
  File "<stdin>", line 1
    def hello(p1 p2):
                  ^
SyntaxError: invalid syntax
```

- An exception is an issue **raise** by the code
- You can choose to **catch** exceptions and so write limit cases of your code
- If the exception is not caught somewhere the code will fail and you will get a **traceback**

And a lot more …

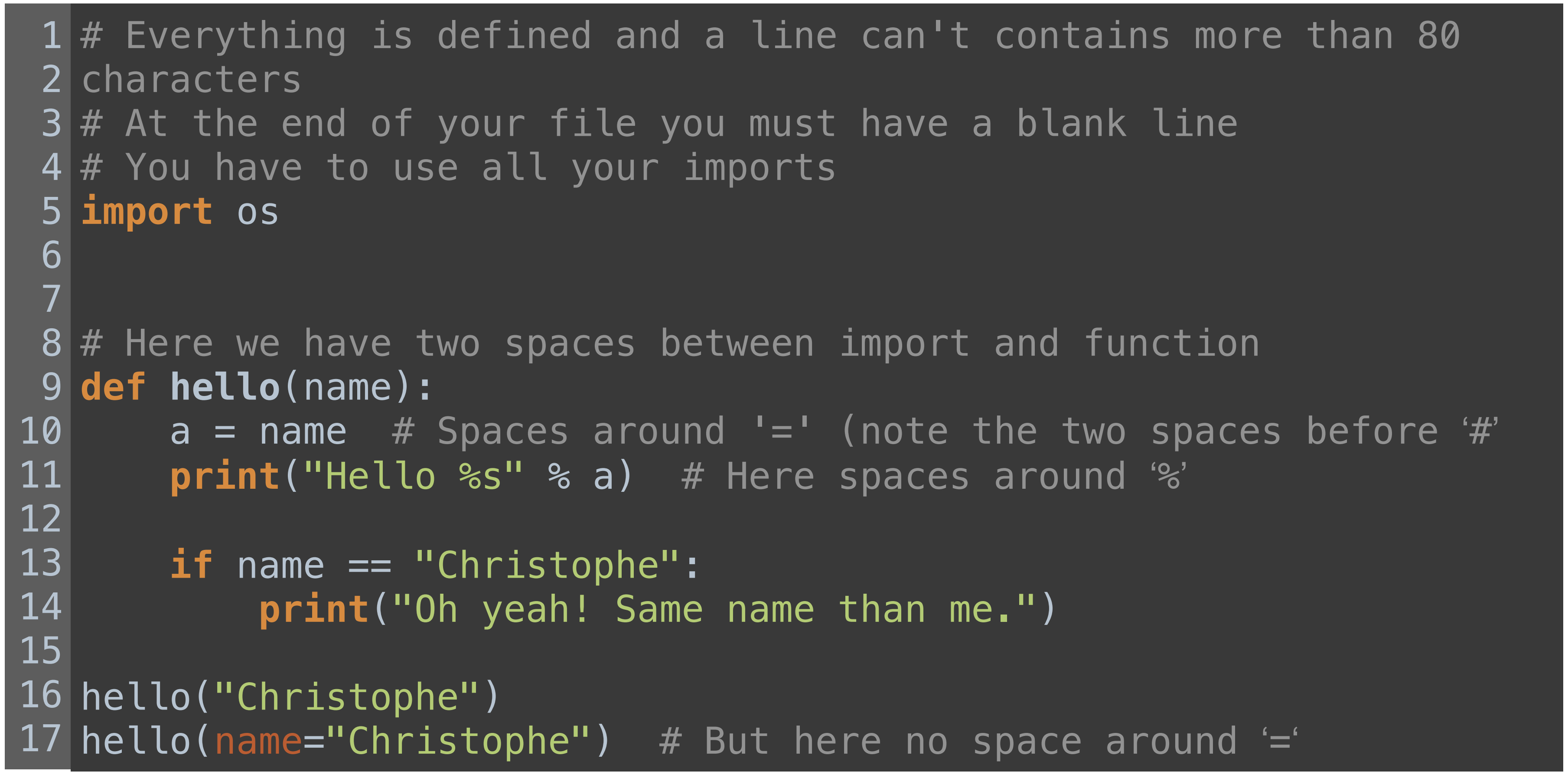
3 Bases: Exceptions



```python
# A try except block is to catch exceptions
# In other languages we call him try/catch block sometimes
try:
    data = [1, 2, 3]
    last = data[4]
except IndexError as er:
    print("Yes! We caugth you babe: %s" % er)
except Exception as er:
    print("Unexpected error happened")
finally:
    # This block code is always executed at the end of try except
    pass
```

- An exception is an issue **raise** by the code
- You can choose to **catch** exceptions and so write limit cases of your code
- If the exception is not caught somewhere the code will fail and you will get a **traceback**

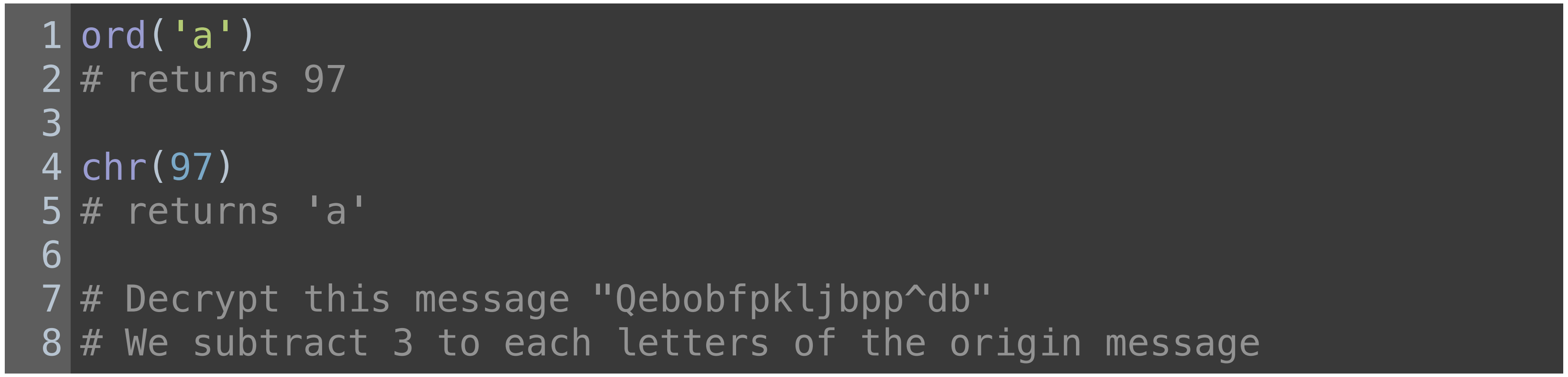
3 Bases: PEP8



```python
# Everything is defined and a line can't contains more than 80
characters
# At the end of your file you must have a blank line
# You have to use all your imports
import os


# Here we have two spaces between import and function
def hello(name):
    a = name  # Spaces around '=' (note the two spaces before ‘#’
    print("Hello %s" % a)  # Here spaces around ‘%’

    if name == "Christophe":
        print("Oh yeah! Same name than me.")

hello("Christophe")
hello(name="Christophe")  # But here no space around ‘=‘
```

- PEP8 is a convention to write clean code and readable by anyone

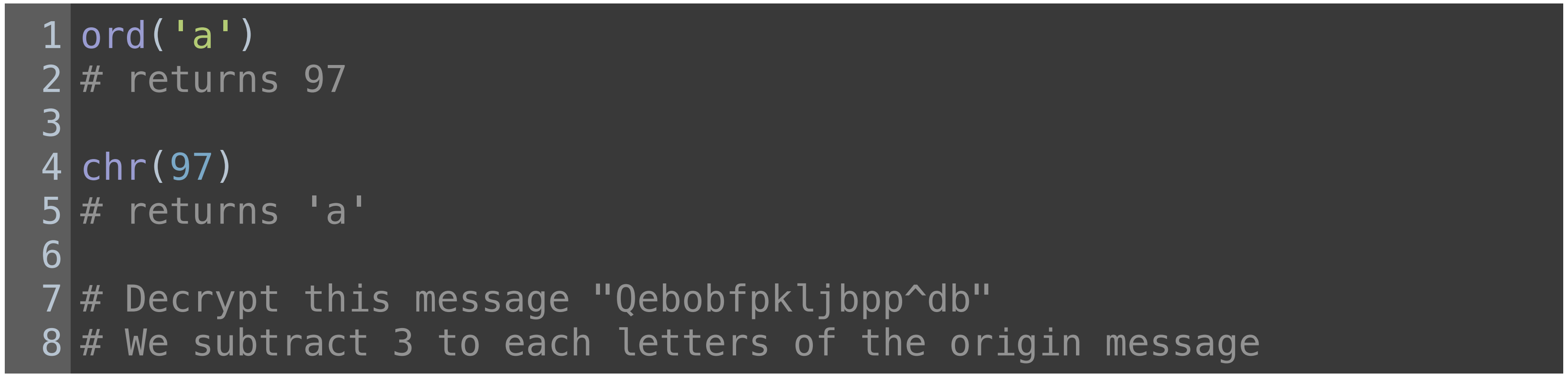
3 Bases: Bonus



```python
ord('a')
# returns 97

chr(97)
# returns 'a'

# Decrypt this message "Qebobfpkljbpp^db"
# We subtract 3 to each letters of the origin message
```

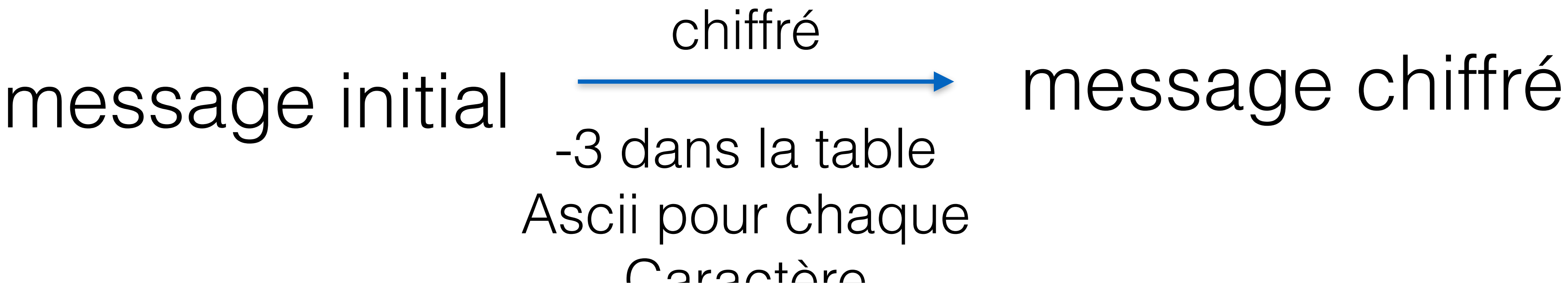
- ord and chr are useful to get character number in the ASCII table

3 Bases: Bonus



```
ord('a')
# returns 97

chr(97)
# returns 'a'

# Decrypt this message "Qebobfpkljbpp^db"
# We subtract 3 to each letters of the origin message
```

- ord and chr are useful to get character number in the ASCII table

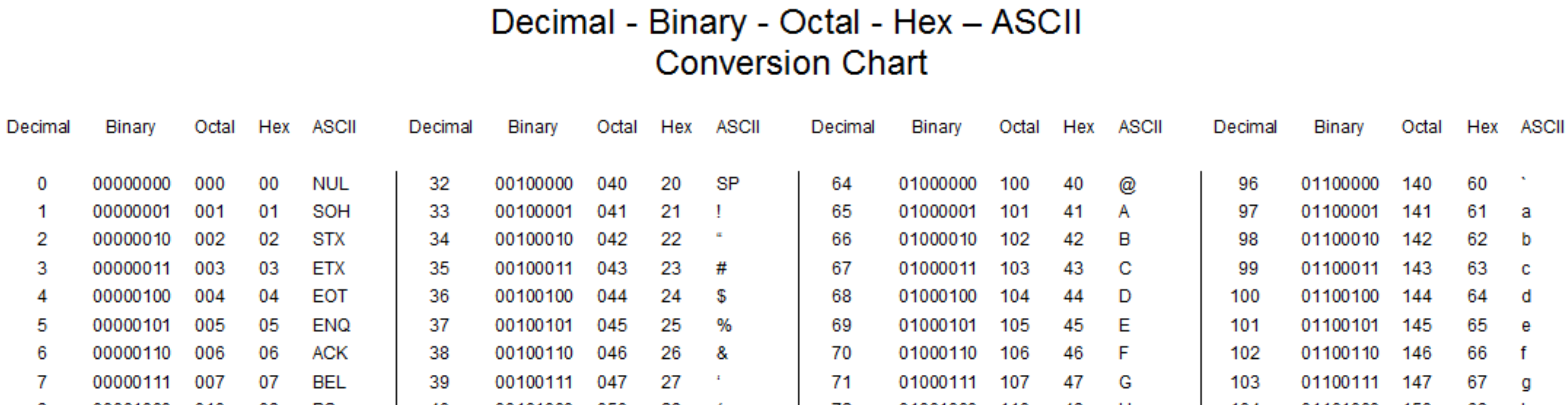


message initial

chiffré

-3 dans la table Ascii pour chaque Caractère

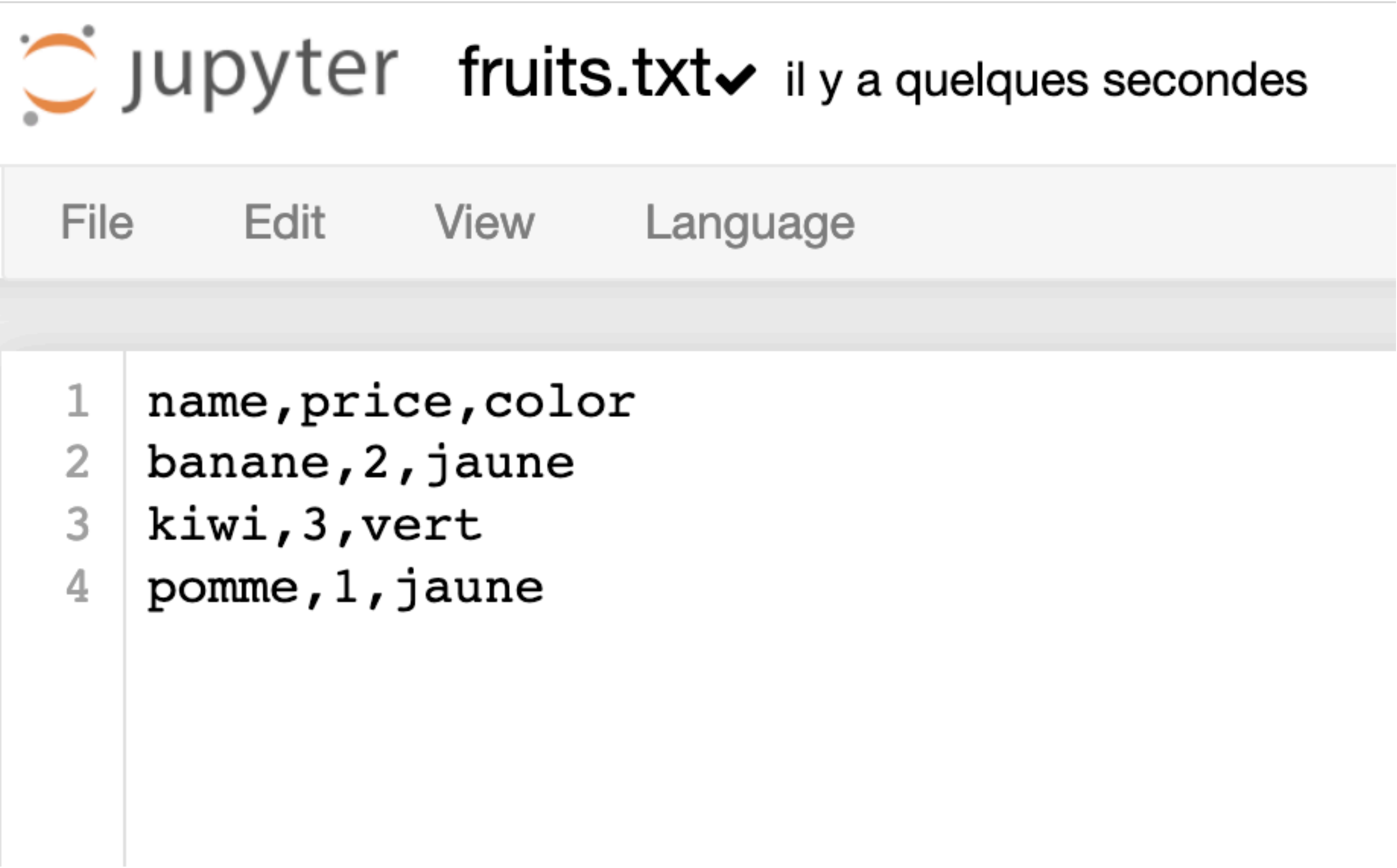
message chiffré



Decimal - Binary - Octal - Hex – ASCII
Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g

3 Bases: Practice #1

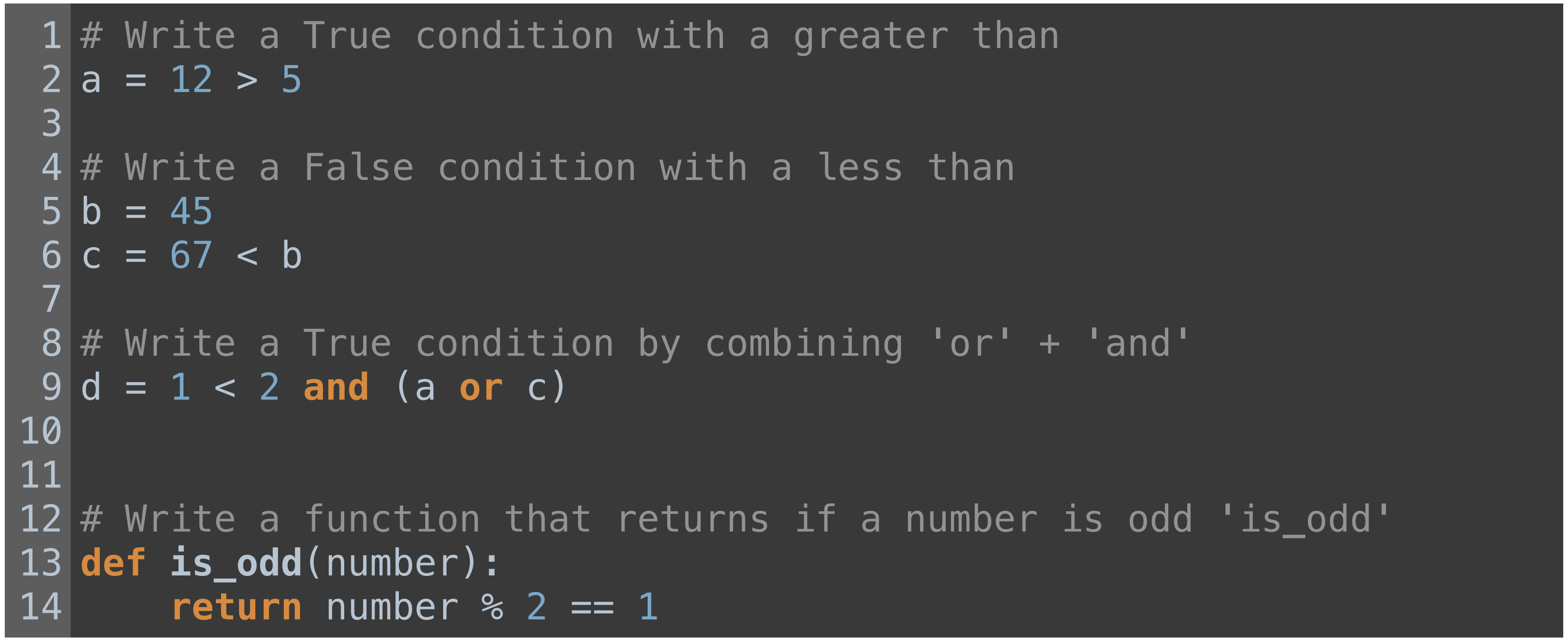


```
name,price,color
banane,2,jaune
kiwi,3,vert
pomme,1,jaune
```

Create a CSV with 3 columns et 4 rows.

1. Open the CSV
2. Create a list of list of data
3. Count with a for #columns
4. Count with a for #rows

3 Bases: Practice #2



```python
# Write a True condition with a greater than
a = 12 > 5

# Write a False condition with a less than
b = 45
c = 67 < b

# Write a True condition by combining 'or' + 'and'
d = 1 < 2 and (a or c)


# Write a function that returns if a number is odd 'is_odd'
def is_odd(number):
    return number % 2 == 1
```

Booleans.

3 Bases: Misc

- Python driven by the indentation in your code
- Don’t hesitate to comment your code
- Code slowly and test at every step your code
- Exceptions and errors in Python are very clear, so **read them please** 😥

4 Hand’s on lab

- We’ll go through:
  - ✓ simple snippets of Python code
  - ✓ algorithm problems
  - ✓ practical data cases
- Google* is your friend if you get stuck
- I’m your friend too

*(maybe solution can be found on Google)

4 Hand’s on lab: Simple programs

1. Write a program that prints a given name

2. Write a function that compares a given date (YYYY-MM-DD) with the present date and returns true if superior and false if not

3. Write a program that lists files in a given directory (use os.walk())

4. Write a *“Rock-Paper-Scissors”* game and play against the computer

4 Hand’s on lab: Algorithms problems

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

pb1 extracted from https://projecteuler.net/

4 Hand’s on lab: Algorithms problems

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Find the largest palindrome made from the product of two 3-digit numbers.

pb4 extracted from https://projecteuler.net/

4 Hand’s on lab: Algorithms problems

Work out the first ten digits of the sum of the following one-hundred 50-digit numbers.

```
37107287533902102798797998220837590246510135740250
46376937677490009712648124896970078050417018260538
74324986199524741059474233309513058123726617309629
91942213363574161572522430563301811072406154908250
23067588207539346171171980310421047513778063246676
89261670696623633820136378418383684178734361726757
28112879812849979408065481931592621691275889832738
44274228917432520321923589422876796487670272189318
47451445736001306439091167216856844588711603153276
70386486105843025439939619828917593665686757934951
62176457141856560629502157223196586755079324193331
64906352462741904929101432445813822663347944758178
92575867718337217661963751590579239728245598838407
58203565325359399008402633568948830189458628227828
80181199384826282014278194139940567587151170094390
35398664372827112653829987240784473053190104293586
86515506006295864861532075273371959191420517255829
71693888707715466499115593487603532921714970056938
54370070576826684624621495650076471787294438377604
53282654108756828443191190634694037855217779295145
36123272525000296071075082563815656710885258350721
45876576172410976447339110607218265236877223636045
17423706905851860660448207621209813287860733969412
81142660418086830619328460811191061556940512689692
51934325451728388641918047049293215058642563049483
62467221648435076201727918039944693004732956340691
15732444386908125794514089057706229429197107928209
55037687525678773091862540744969844508330393682126
18336384825330154686196124348767681297534375946515
80386287592878490201521685554828717201219257766954
78182833757993103614740356856449095527097864797581
16726320100436897842553539920931837441497806860984
48403098129077791799088218795327364475675590848030
87086987551392711854517078544161852424320693150332
59959406895756536782107074926966537676326235447210
69793950679652694742597709739166693763042633987085
41052684708299085211399427365734116182760315001271
(…)
```

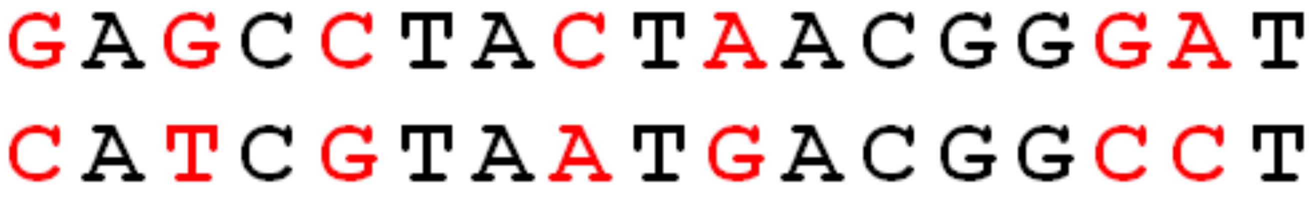
pb13 extracted from https://projecteuler.net/

4 Hand’s on lab: Algorithms problems

Given two strings *s* and *t* of equal length, the **Hamming distance** between *s* and *t*, denoted *dH(s, t)*, is the number of corresponding symbols that differ in *s* and *t.*

Given: Two DNA strings s and t of equal length

Return: The Hamming distance *dH(s, t)*



```
GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT
```

This input should returns a distance of 7

4 Batman wants to escape

https://www.codingame.com/ide/puzzle/shadows-of-the-knight-episode-1

A Resources

- Python website: https://www.python.org/
- Codecademy: https://www.codecademy.com/tracks/python
- Python 3 docs: https://docs.python.org/3/contents.html
- Project Euler: https://projecteuler.net/
- CodeWars: https://www.codewars.com/
- Coding Game: https://www.codingame.com/
- http://dataquest.io

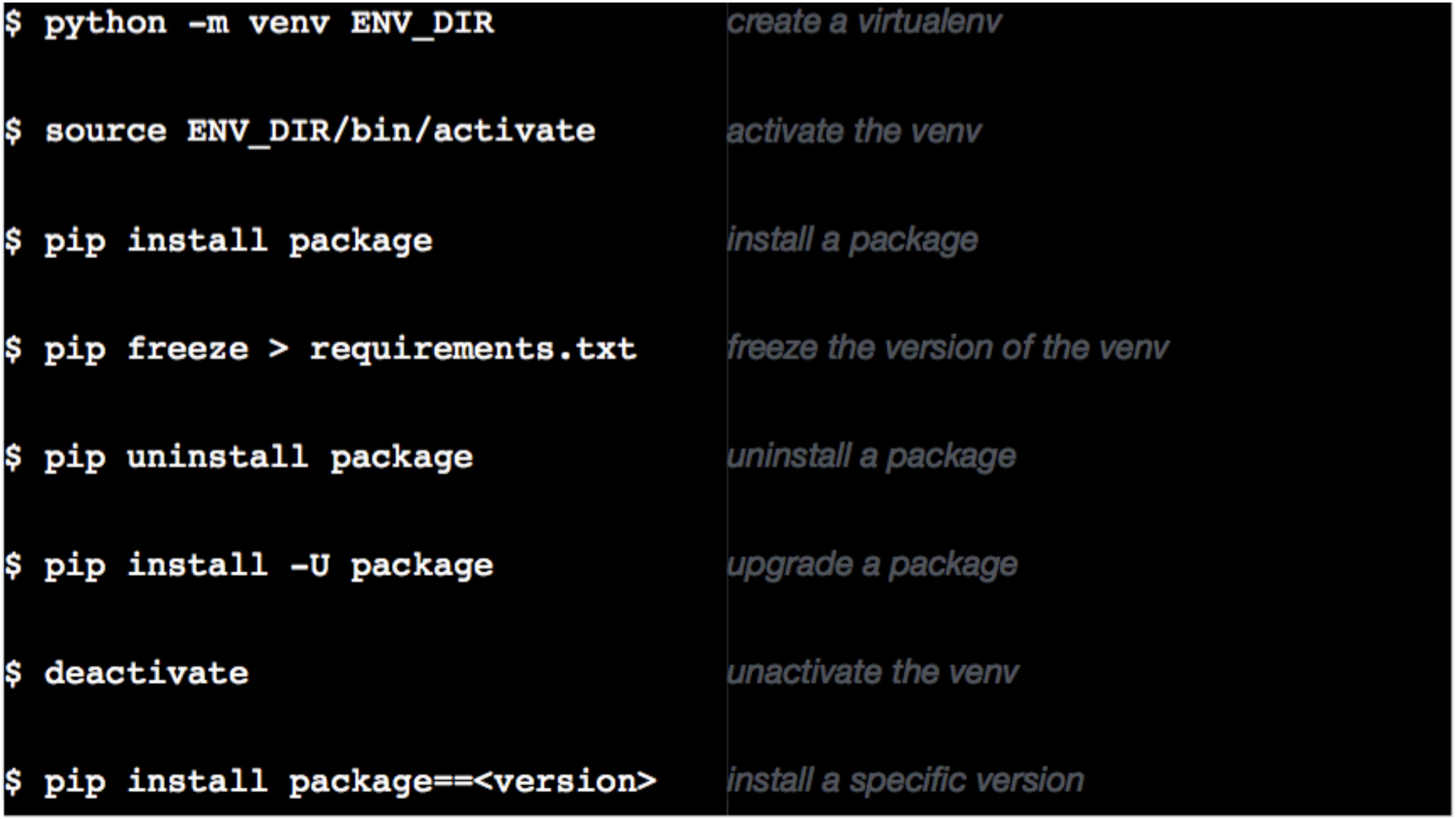
B Course index

- Variables
- Types
- Types operations (int + float)
- Types operations (list)
- Types operations (str)
- Types operations (dict)
- Loops
- If statements
- Types operations (list) - advanced
- Functions
- Main block
- Open and read files
- Open and write files
- Imports
- datetime
- Exceptions
- PEP8

C How to version your code with Git

- If you are on linux install git (apt-get install git)
- On windows go to: https://git-scm.com/downloads

D Use pip // virtualenvs



`$ python -m venv ENV_DIR`	*create a virtualenv*
`$ source ENV_DIR/bin/activate`	*activate the venv*
`$ pip install package`	*install a package*
`$ pip freeze > requirements.txt`	*freeze the version of the venv*
`$ pip uninstall package`	*uninstall a package*
`$ pip install -U package`	*upgrade a package*
`$ deactivate`	*unactivate the venv*
`$ pip install package==<version>`	*install a specific version*

0 Thanks a lot!