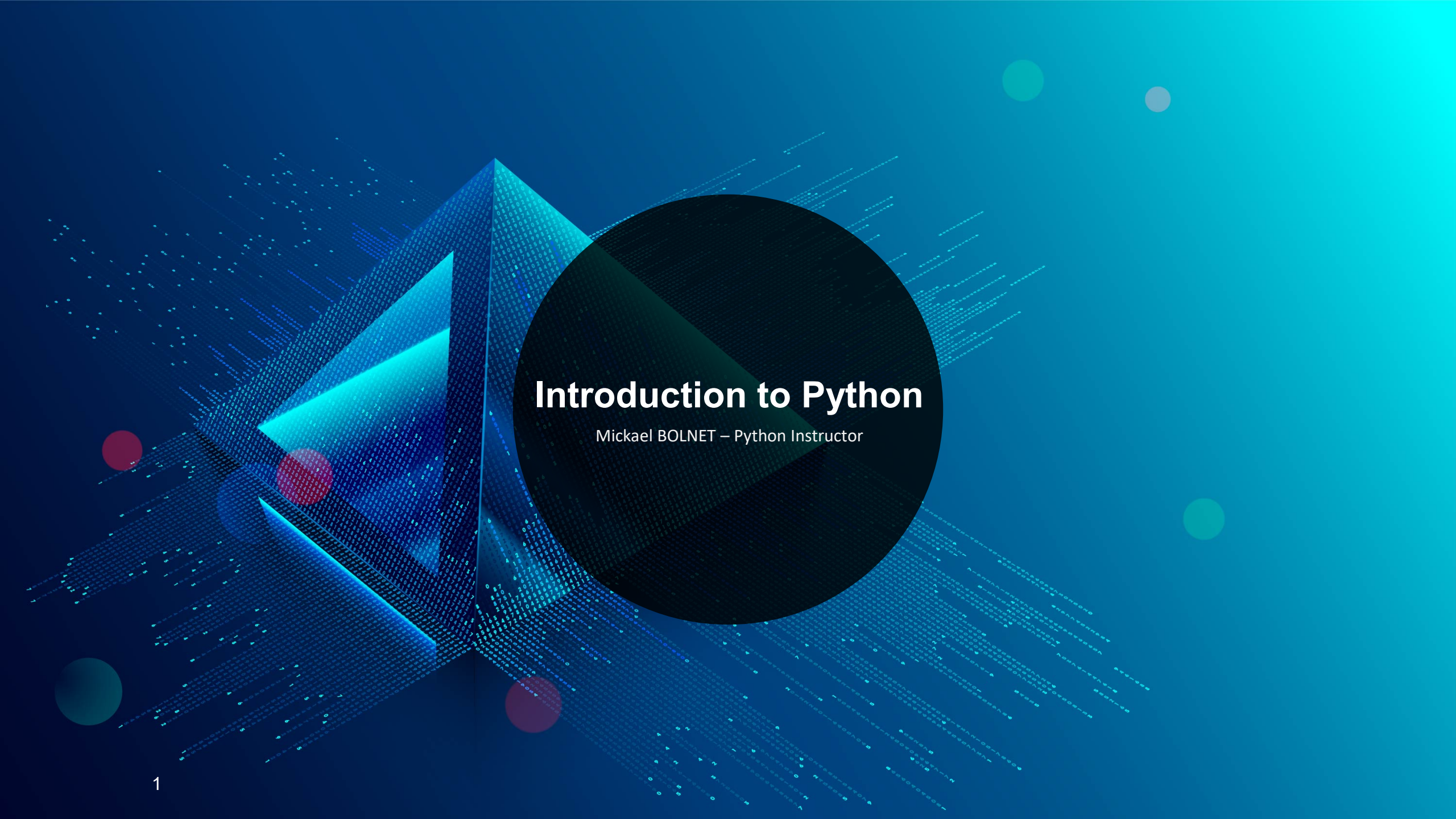
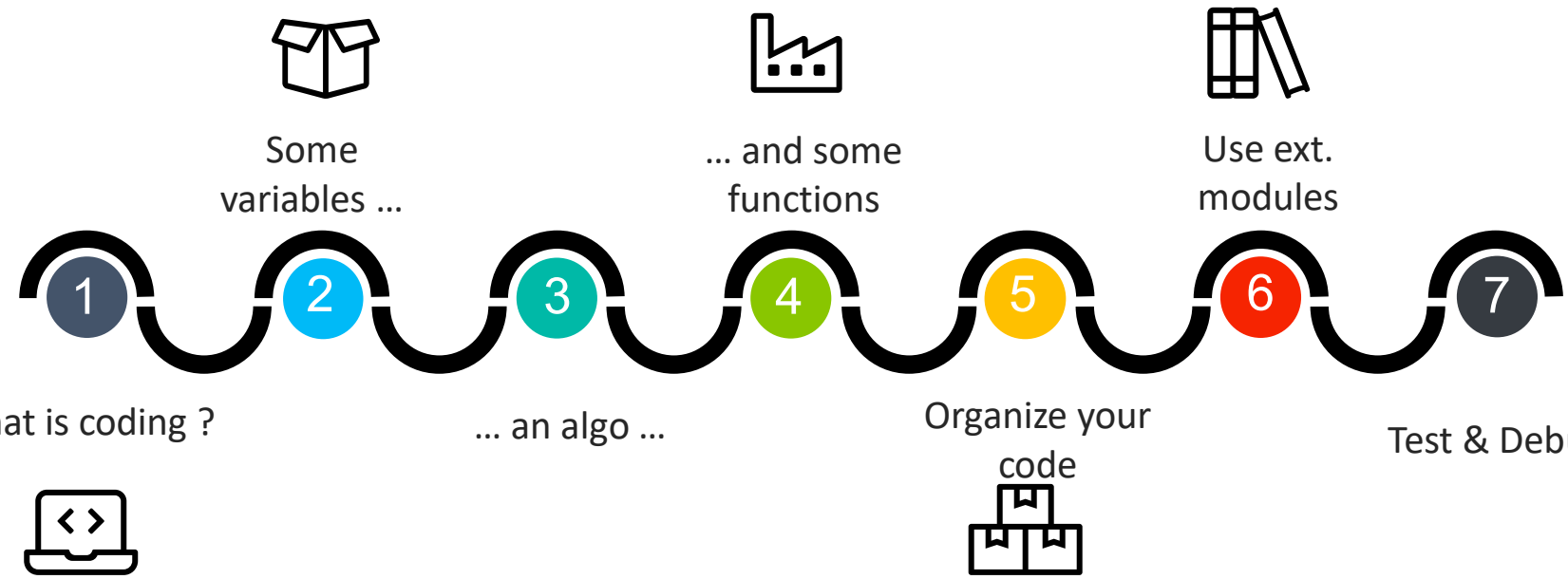

Introduction to Python
Mickael BOLNET – Python Instructor

Introduction to Python

Table of contents



Guido Van Rossum

Creator Of Python - since 1989
From the Netherlands
Python version 1.0 - 1991
Python Version 2.0 - 2000…2015 (err… 2020)
Python version 3.0 - 2008

How does it work ?

Memory

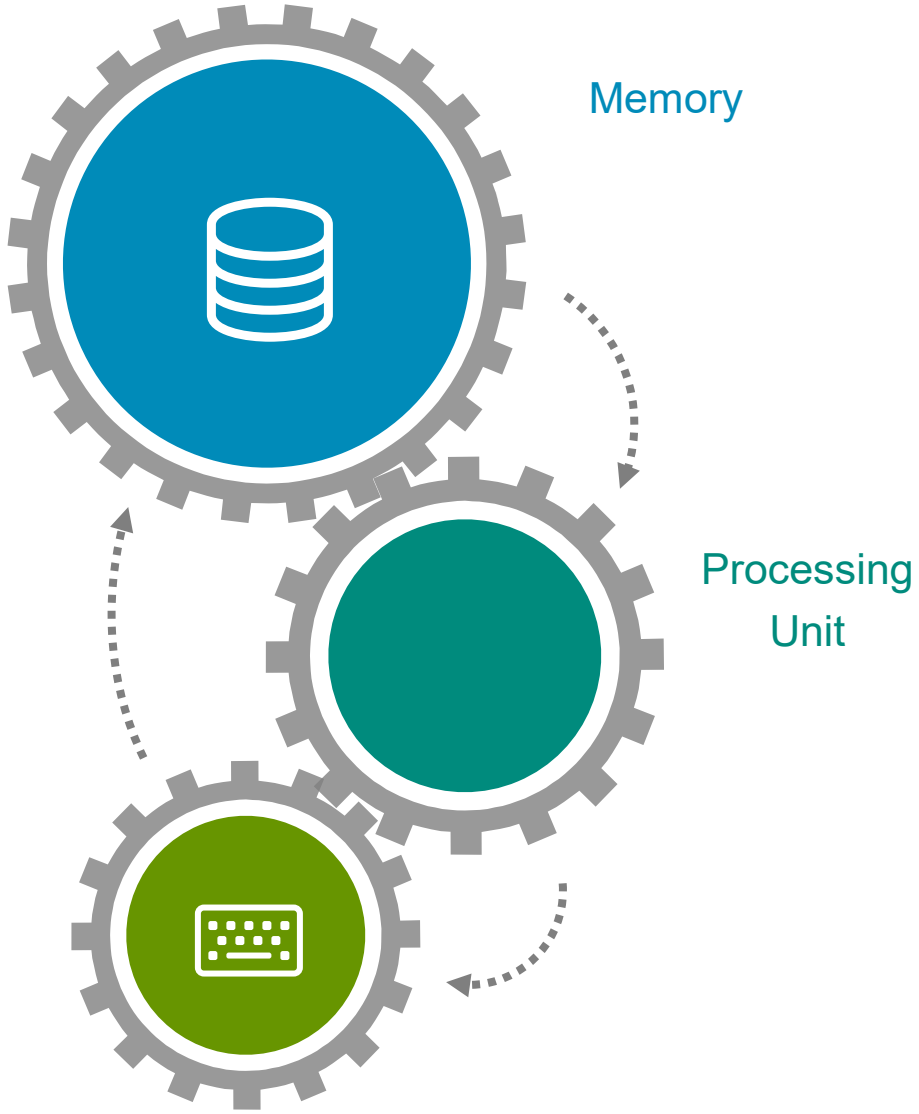
Like in a human being memory is a important part of a computer.
This is where the code describing what the computer should do
Is stored. And this is where data is stored.

Processing Unit

One or many processing units execute the code.
There are many types of processors.
CPU / GPU / TPU

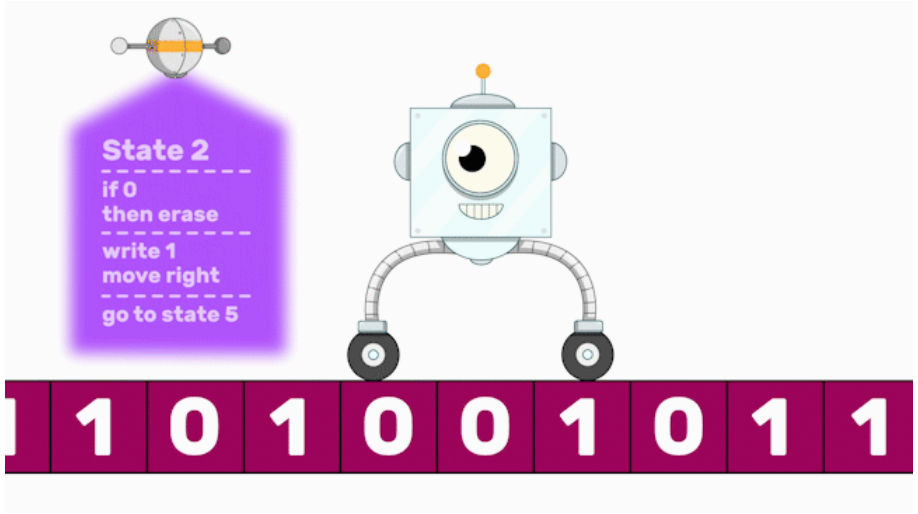
Périphériques

Peripherals allows the system to interact with the world.
Keyboard, Screen, mouse, network…

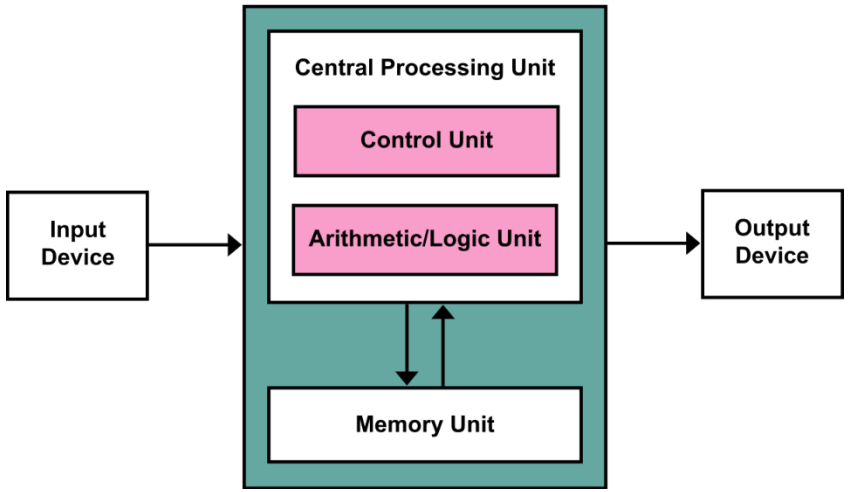


How does it work ?

Turing Machine



Von Neumann Architecture



How does it work ?

Everything start with the l'assembly…

```
00000000    push    ebp
00000001    mov     ebp, esp
00000003    movzx   ecx, [ebp+arg_0]
00000007    pop     ebp
00000008    movzx   dx, cl
0000000C    lea     eax, [edx+edx]
0000000F    add     eax, edx
00000011    shl     eax, 2
00000014    add     eax, edx
00000016    shr     eax, 8
00000019    sub     cl, al
0000001B    shr     cl, 1
0000001D    add     al, cl
0000001F    shr     al, 5
00000022    movzx   eax, al
00000025    retn
```

Python in comparison

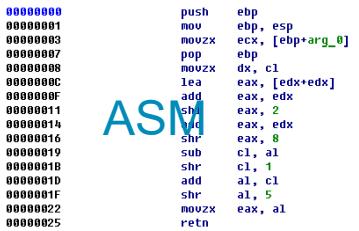
Assembly

C

C++

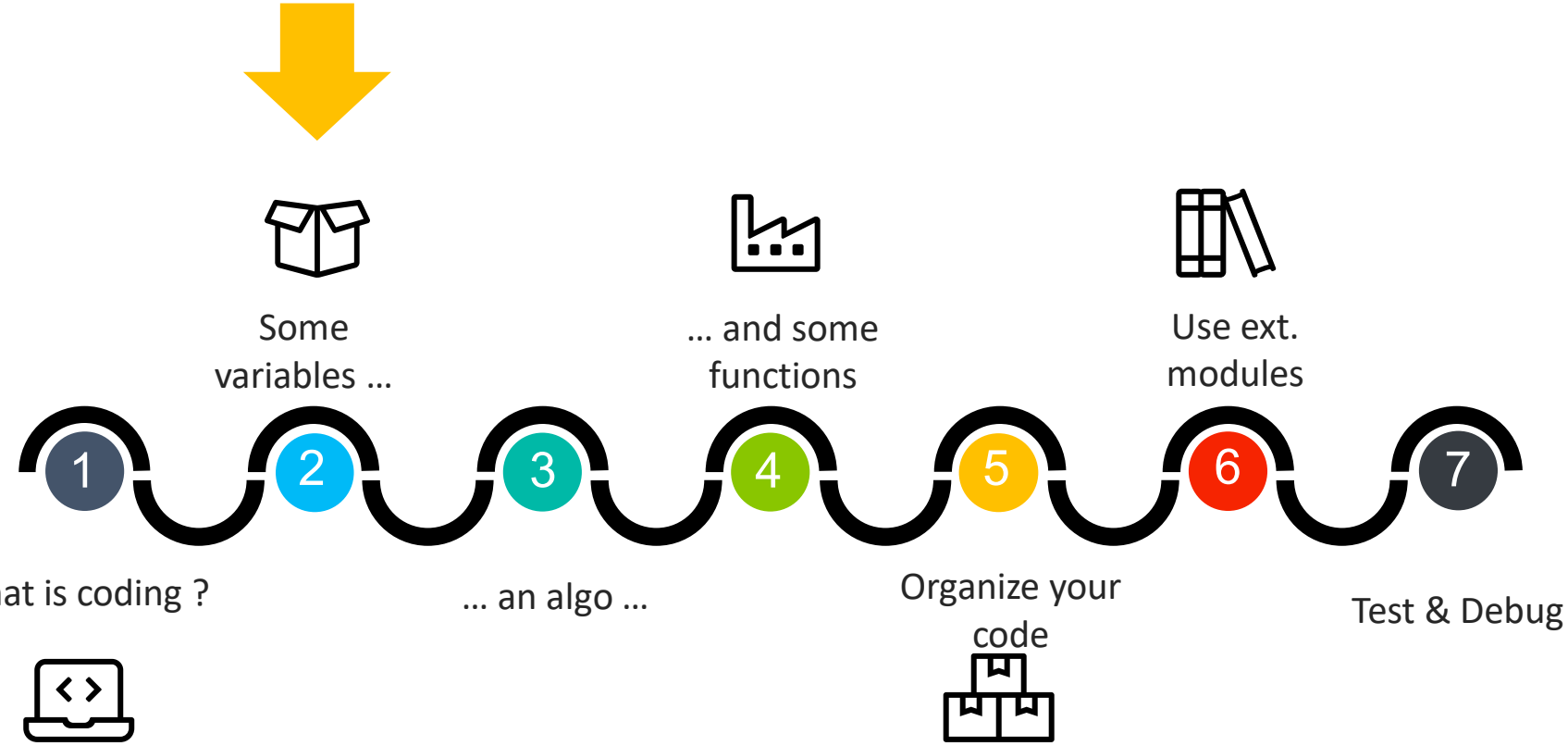
Python

JavaScript



Python use cases

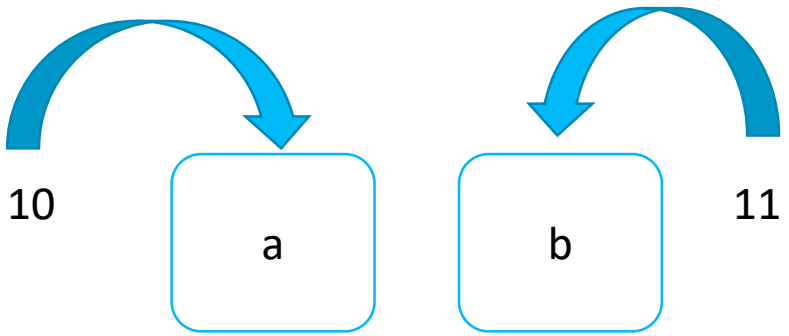
Some variables



Variables

Variables, naming conventions, types (dynamic types), sequences

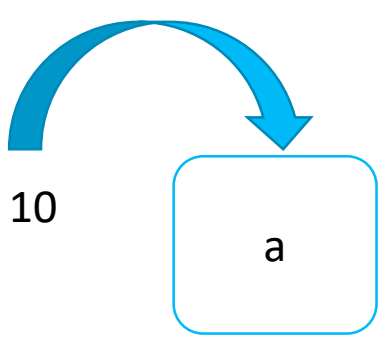
```
a = 10 # 10 → a
b = 11 # 11 → b
print(a) # display the content of a
print("a") # display the character "a"
print(b) # display the content of b
print(a + b) # display the result of an operation
print("a + b") # display the string "a+b"
```



Variables

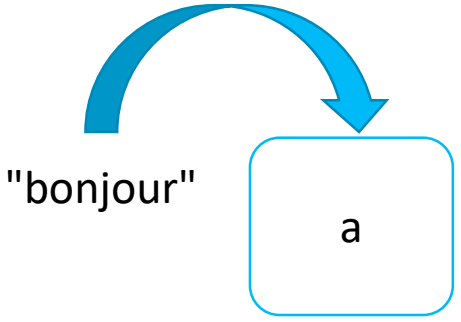
Variables, naming conventions, types (dynamic types), sequences

```
a = 10 # 10 → a
print(type(a)) # display the type of variable a
a = "hello" # "hello"→ a
print(type(a)) # display the type of variable a
a = True # True → a
print(type(a)) # display the type of variable a
```



type(a) → int

Puis…



type(a) → str

Variables

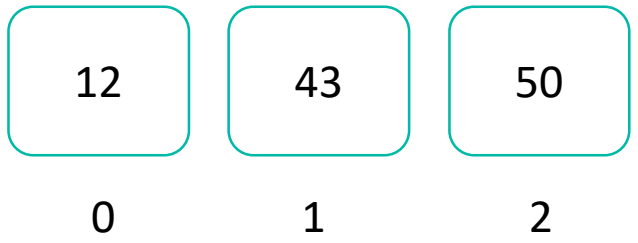
Lists

Entrée [ ]:

```
price_list = [12, 49, 52] # Create a list of 3 integers
print(type(price_list)) # the type of price_list is list

random_list = [True, 12, "50", [1, 4, 2]] # Create a list of random elements
print(type(random_list)) # the type of random_list is list

print(price_list[0]) # prints the first element in price_list
print(price_list[1]) # prints the second element in price_list
print(price_list[2]) # prints the third element in price_list
print(price_list[-1]) # prints the last element in price_list
```



Variables

Lists

Entrée [ ]:

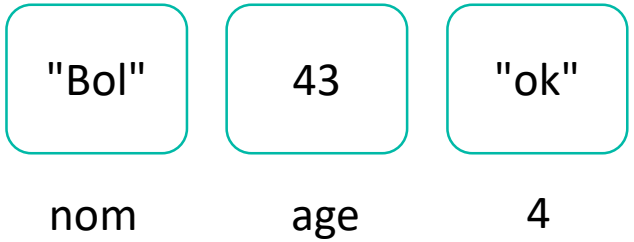
```
price_list = [12, 49, 52, 2, 14, 6, 0] # Create a list of 7 integers

print(price_list[0:4]) # prints elements from the first to the 4th
print(price_list[:4]) # prints elements from the first to the 4th
print(price_list[2:]) # prints elements from the third to the end
print(price_list[2:-1]) # prints elements from the third to the second to last
print(price_list[2::2]) # prints every other elements from the third to the end
```

Variables

Dictionary

Entrée [4]:
```python
personne = {"nom": "Bol", "age": 43, 4: "ok"}
print(personne["nom"]) # -> affiche "Bol"
print(personne[4]) # -> affiche "ok"
print(personne[0]) # n'existe pas
```



Variables

	Mutable	Hashable	Iterable	Indexable	Sliceable
Types bases		x			
List	x		x	x	x
Dictionnary	x		x	x	
Tuple		x	x	x	x
Set	x		x		
String		x	x	x	x

- Hash() return the hash of an hashable
- In order to define an **iterable** one should define the method __iter__ which returns a list or a generator
- In order to define an **hashable** one should define the method __hash__ which returns an unique hash
- In order to define an **indexable** one should define the method __getitem__(self,idx)
- In order to define an **sliceable** one should define the method __getitem__(self,slice)

Les variables

Operations on integer, float and complex

x + y	Addition
x - y	Soustraction
x * y	Multiplication
x / y	Division
x // y	Euclidian Division
x % y	Remainder
-x	
+x	
x ** y	Power

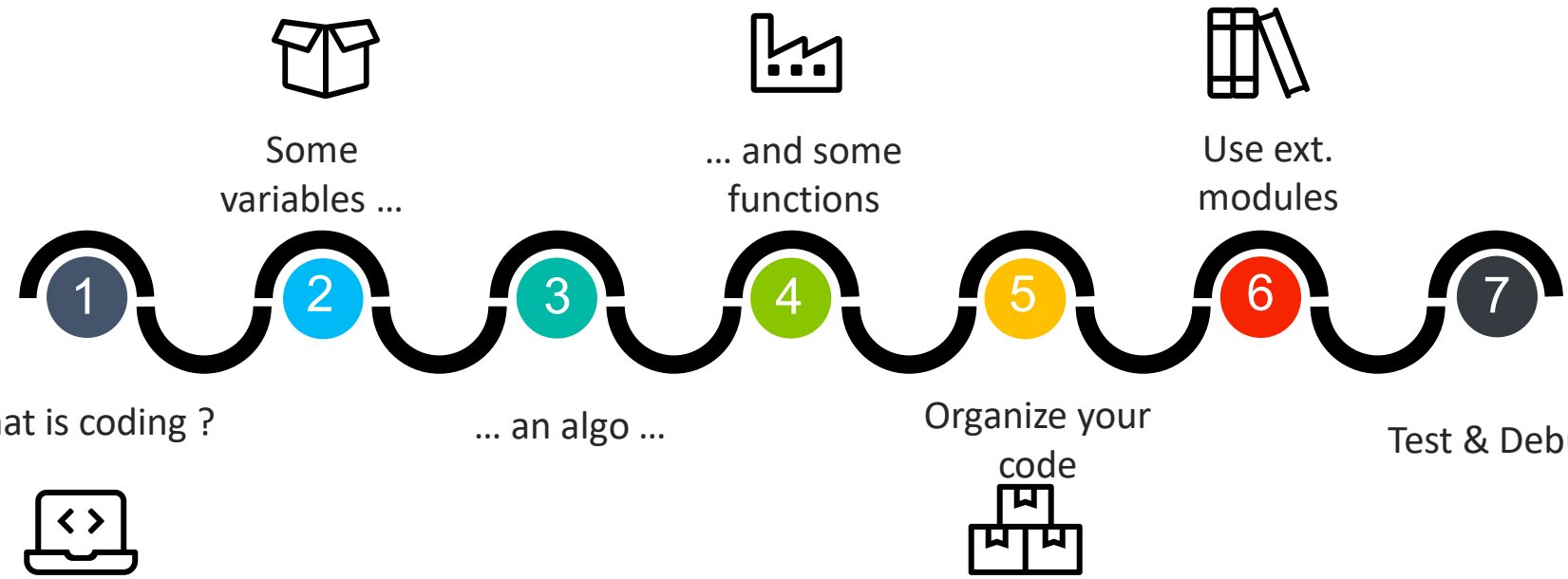
Les variables

Operation on sequences

x not in s	False if s contains x, else True
s1 + s2	Concatenation
s * n	Repeat
s[i]	The i-th element
len(s)	Sequence length
min(s)	Smallest element in seq
max(s)	Biggest element in seq
s.index(x)	Index of x’s first occurence
s.count(x)	Number of x’s occurrences

An Algo…

How to build an algo ?



An algo

WHILE loop

```
nb = 7
i = 0

while i < 10:
    print(i + 1, "*", nb, "=", (i + 1) * nb)
    i += 1
```

Beware : indentation required!

An algo

FOR loop

```
for i in range(5):
    print(i)

for i in range(3, 6):
    print(i)

for i in range(4, 10, 2):
    print(i)

for i in range(0, -10, -2):
    print(i)
```

An algo

Conditionnal structure

```
name = 'Mickael'
if name == 'Mickael':
    print(‘Hello Mickael')
elif name == 'Laetitia':
    print(‘Hello Laetitia')
else:
    print(“You can’t get inside ”)
```

Beware double "=" !!

An algo

Comparison operators

<	Less than
>	Greater than
<=	Less or equal
>=	Greater or equal
==	Equal
!=	Different

An algo

Combining operators

- or
- not
- and

An algo

Time for Fizz Buzz !!

Another example of algorithm

Bubble sort

Inputs and templates

Interact with user

```
name = input(" What is your name ? ")
age = int(input(" How old are you ? ")) # cast string value to int

"  Variable : %type" % var
"  Variables : %type, %type" % (var1, var2)
"Result : %(val)type %(unit)type" % {'val':var1, 'unit':var2}

type is d : integer - f : float - s : string - c : character - o : octal - x : hexadecimal

Precision pour float :
• "Result: %.2f" % 3.141592653589793
```

F-string for python 3

Interact with user

```
name = input(" What is your name ? ")
age = int(input(" How old are you ? "))

print(f" Je m’appelle {name} et j’ai {age} ans" )
```

Dichotomy search

Functions

Definition vs Execution

```
def say_hi():
    print(‘Hello world!’)


say_hi() #’Hello world!'
```

Functions

Parameters

```
def say_hi():
    print(‘Hello world!')

def say_hi(name):
    print(' Hello  ' + name)

def say_hi(name, name2='', name3='toto'):
    print('Hello ' + name + ' ' + name2)
```

Functions

Portée des variables (local vs global)

```
foo = 1
def test_local():
    foo = 2 # new local foo
def test_global():
    global foo
    foo = 3 # changes the value of the global foo
```

Board display

External modules

Pip, parenthèse environnement, Selenium

Install modules in the command line interface

pip install selenium

Create a list of installed external modules

pip freeze > requirements.txt

Install a list of external modules

pip install –r requirements.txt

Create a virtual env

virtualenv monenv

Activate / Deactivate environment

monenv/Scripts/activate

deactivate

Example with Selenium

Organize your code

Starts with:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

Contains __init__.py :

```
MyPackage/
        __init__.py
        MyModule.py
        MyModule2.py

__all__ = [ 'MyModule', 'MyModule2']
```

Organize you code

Packages & Modules

```
import MyModuleLibrary.MyModule
import MyModuleLibrary.MyModule2

MyModuleLibrary.MyModule.function_welcome()
MyModuleLibrary.MyModule2.function_welcome_bis()
```

Package your module


```
setup.py
src/
    mypkg/
        __init__.py
        module.py
        data/
            tables.dat
            spoons.dat
            forks.dat
```

Package your module


```
#!/usr/bin/env python

from distutils.core import setup

setup(name='MonPackage',
      version='1.0',
      description='Package that says hello in several languages',
      author='Mickael BOLNET',
      author_email='mickael.bolnet@web-n-data.com',
      packages=['MonPackage'],
      requires=['numpy'],
      package_dir={'MonPackage': 'src/MonPackage'},
      package_data={'MonPackage': ['data/*.dat']},
    )
```

Get arguments from cmdline

Argparse

```
parser = argparse.ArgumentParser(
        description="This script does something.")
parser.add_argument("who", help="Who are you ?")
parser.add_argument("-m","--many", type=int)
args = parser.parse_args()
for i in range(args.many):
        print("Hello " + args.who)
```

Test & Debug

```
def add(a, b):
    """
        :Example:
        >>> add(1, 1)
        2
        >>> add(2.1, 3.4)
        5.5
    """
    return a + b


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Test & Debug

```
import unittest
from training.poo.bank import bank
class TestDeposit(unittest.TestCase):
    def setUp(self):
        self.account = bank.BankAccount('012345', 500)
    def testBasicDeposit(self):
        self.account.deposit(100)
        self.assertEqual(600, self.account.balance())
    def tearDown(self):
        del self.account
```

Test & Debug

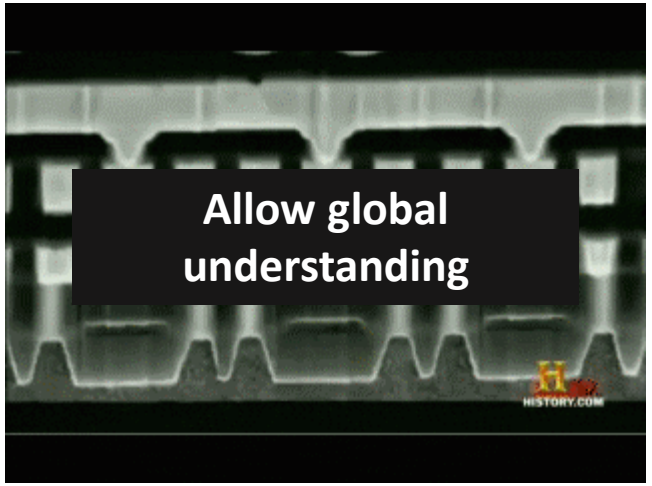
- l : (list) list some line of code around
- n : (next) execute next line
- s : (step in) get inside function execution
- r : (return) get outside function execution
- unt : (until) execute a loop untill the end
- q : (quit) quit program execution
- c : (continue) continue untill next breakpoint

Sphynx & reStructuredText

Object Oriented Programming

OOP


Avoid long linear code


Allow global
understanding

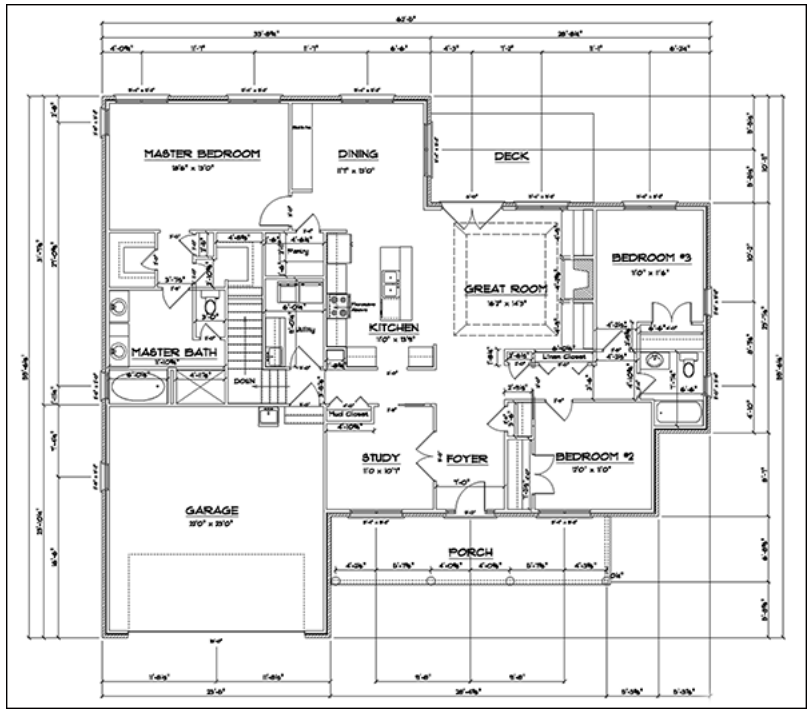

Split responsabilities
Working together

OOP

An object is defined by

- Its states -> attributes
- Its behaviours -> methods

Class vs Instance



Classes are blueprints

Instances are real

OOP

```
class Person:
"""Class that define a person by:
   - its firstname
   - its lastname
   - its age
   - its location"""


   def __init__(self):
      """ The constructor describe how to build each instance. So far every person have the same characteritics """
      self.firstname = "John"
      self.lastname = "Smith"
      self.age = 33
      self.location = "Paris"
```

OOP

```
person= Person()
print(person.firstname)
```

OOP

```
class Person:
"""Class that define a person by:
   - its firstname
   - its lastname
   - its age
   - its location"""



   def __init__(self, firstname, lastname="John", age=32):
      """ The constructor describe how to build each instance. So far every person have the same characteritics """
      self.firstname = firstname
      self.lastname = lastname
      self.age = age
      self.location = "Paris"
```

Attention à ne PAS OUBLIER self !

OOP

```
person= Person("Martin","Jean")
print(person.firstname)
```

OOP

```
class Person:
"""Class that define a person by:
   - its firstname
   - its lastname
   - its age
   - its location"""

   nb_person = 0 # initialize a class attribute (common for every Person)
   def __init__(self, firstname, lastname="John", age=32):
      """ The constructor describe how to build each instance. So far every person have the same characteritics """
      self.firstname = firstname
      self.lastname = lastname
      self.age = age
      self.location = "Paris"
      Person.nb_person += 1 # add one to nb_person every time we create a new instance
```

Attention à ne PAS OUBLIER self !

Special methods

- __init__(self) : initializer, called after instance creation
- __del__(self) : destructor called before instance destruction
- __str__(self) -> str : Is called to convert instance to string (i.e. for print or str cast). It must return a string

Special methods

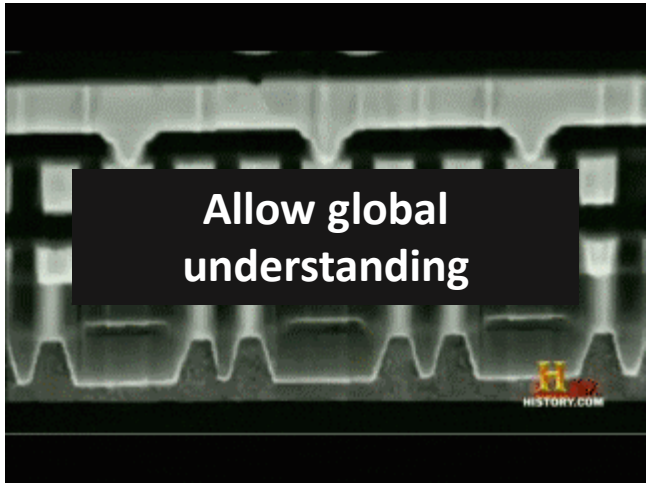
Method	Operator
__lt__(self, other)	x < y
__le__(self, other)	x <= y
__eq__(self, other)	x == y
__ne__(self, other)	x != y
__ge__(self, other)	x >= y
__gt__(self, other)	x > y

Special methods

Method	Operator
__neg__	-x
__add__	x + y
__sub__	x - y
__mul__	x * y
__div__	x / y

OOP


Avoid long linear code


Allow global
understanding


Split responsabilities
Working together

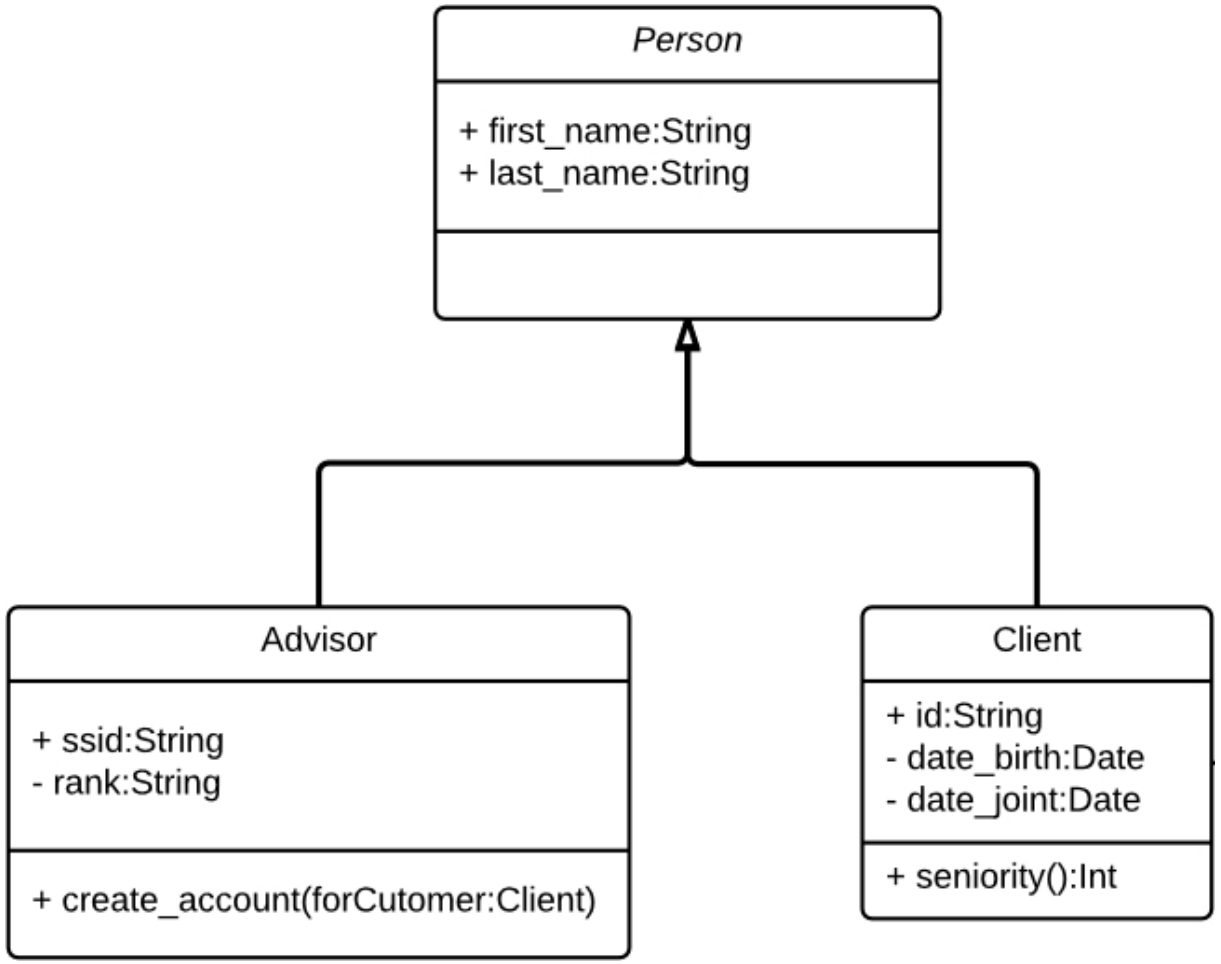
OOP

An object should be a black box

Only the developer in charge knows what is inside

- Make internal changes independant from external use
- Manage complexity internally
- More robust development

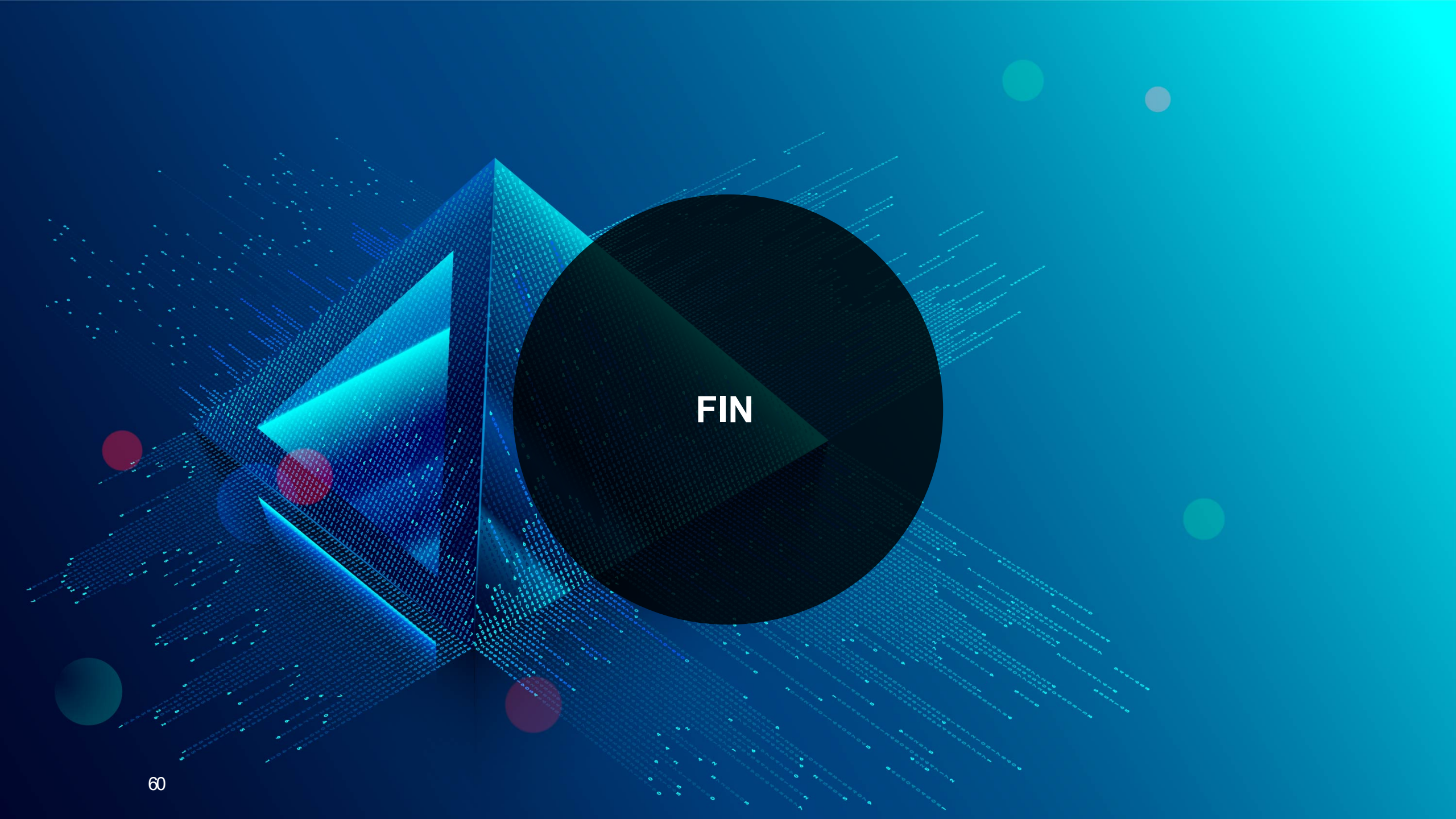
Inheritance

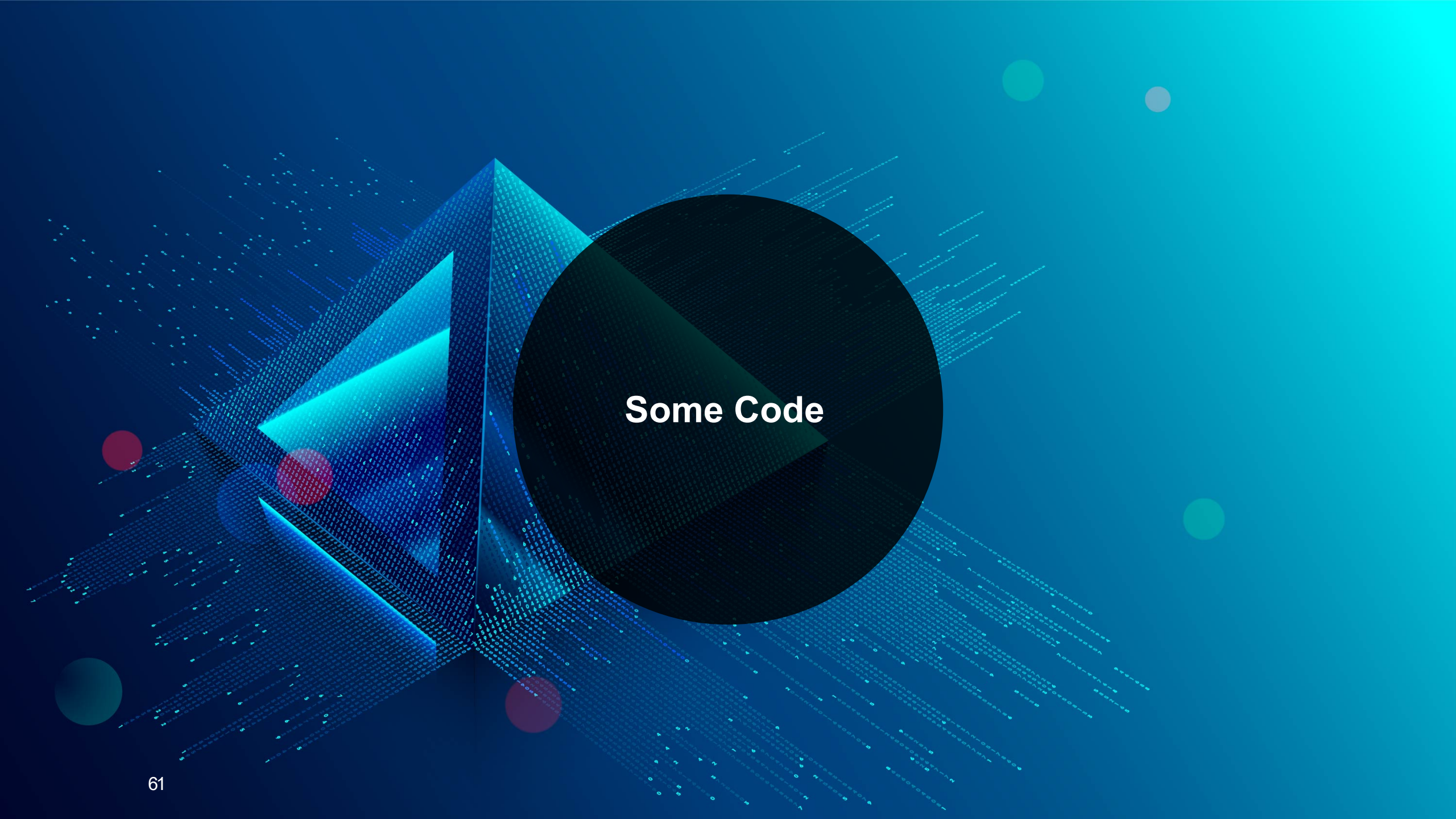


Inheritance

```
class Client(Personne):
"""Classe définissant une personne caractérisée par :
   - son nom
   - son prénom
   - son âge
   - son lieu de résidence"""


   def __init__(self):
      """Pour l'instant, on ne va définir qu'un seul attribut"""
      Personne.__init__(self)
```


FIN


Some Code