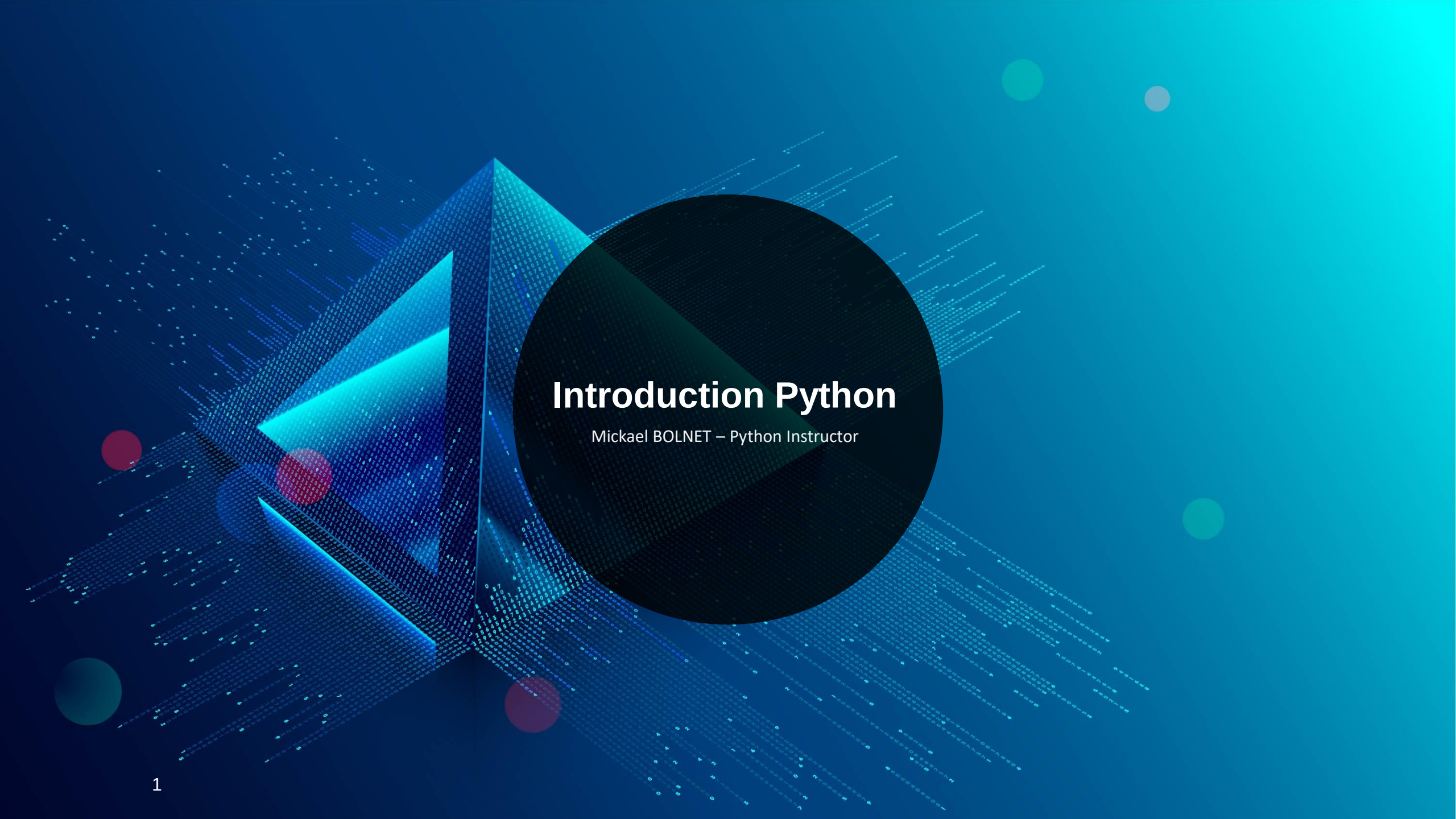
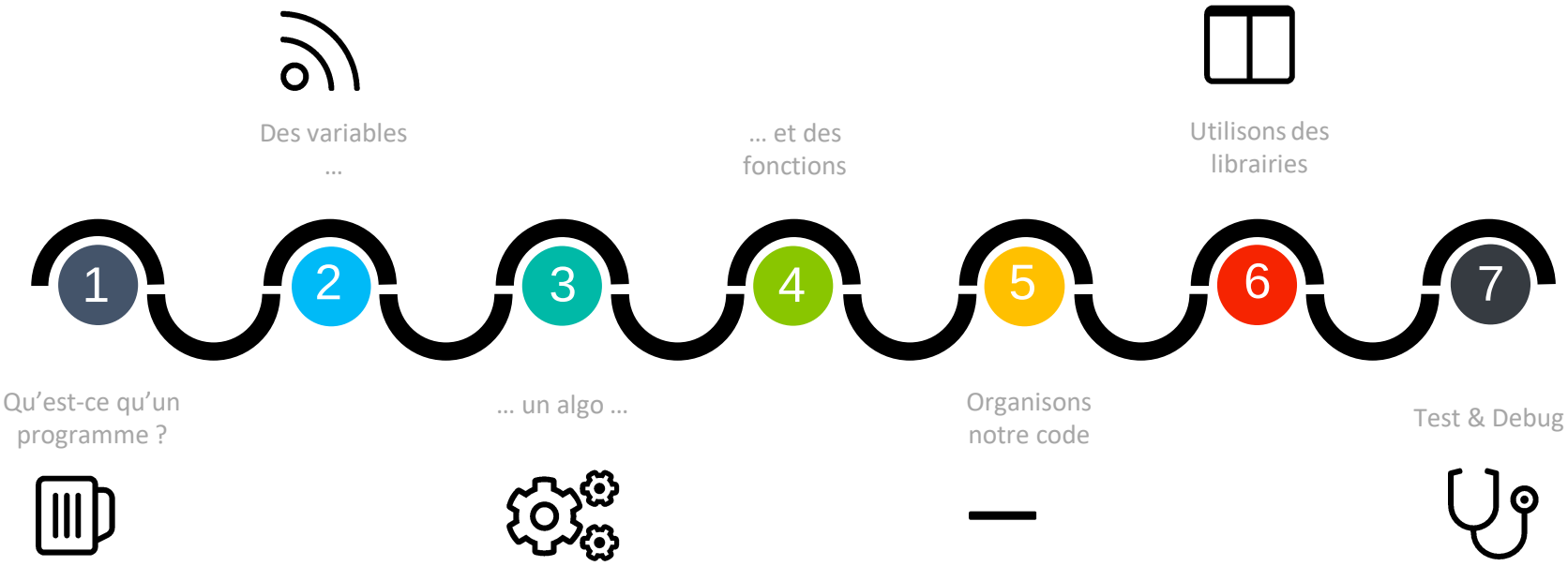

Introduction Python
Mickael BOLNET – Python Instructor

Introduction Python

Programme de la formation



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

Comment ça fonctionne ?

Put a relevant subtitle in this line

Mémoire

La mémoire est comme pour l'être humain un élément de essential dans le fonctionnement d'un ordinateur. Le code qui décrit le comportement d'un algorithme y est stocké. Ainsi que les données nécessaires.

Processing Unit

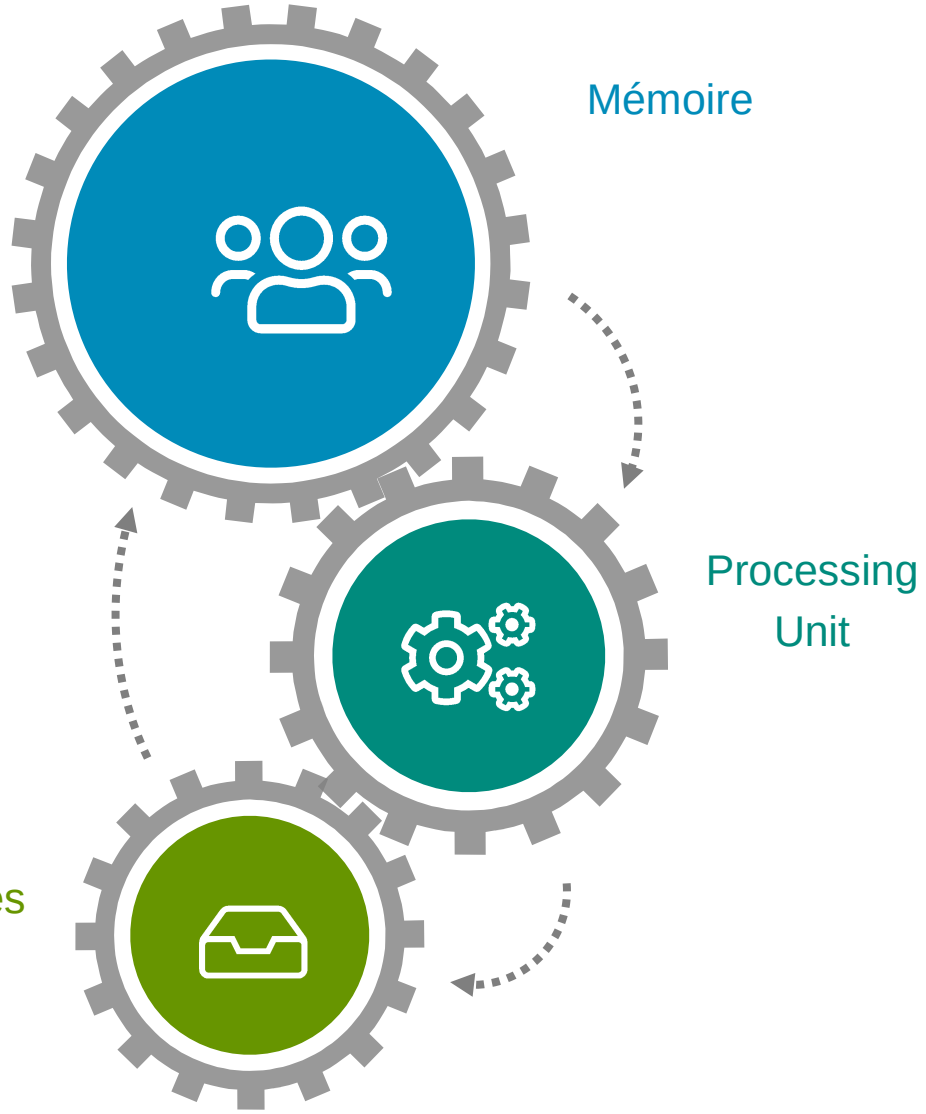
Le ou les processeurs sont les uniités en charge d'executer la logique du programme.

On trouve plusieurs types de processeurs CPU/GPU/TPU…

Périphériques

Les peripheriques permettent de gérer les interactions avec l'extérieur.

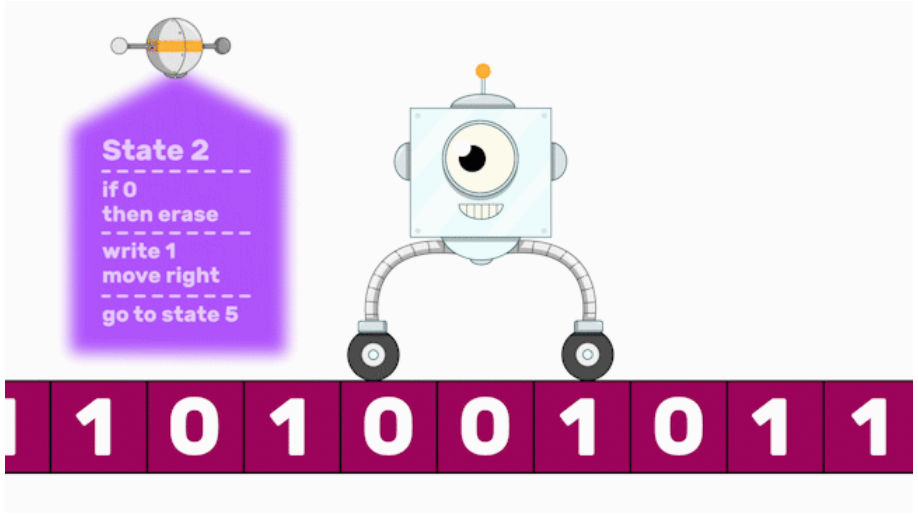
Clavier, écran, souris, reseaux…



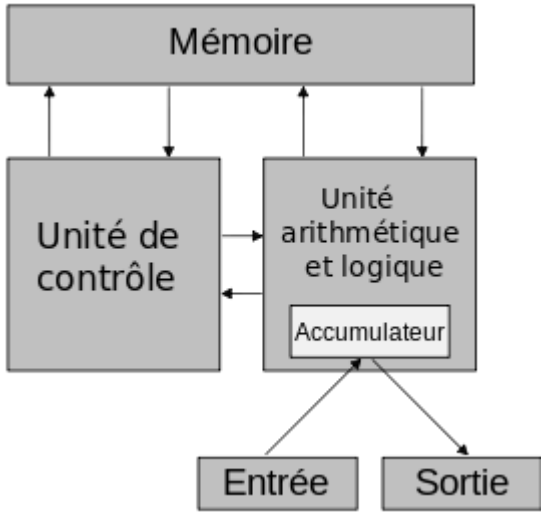
Qu’est-ce qu’un programme ?

Les machines séquentielles, machine de Turing, architecture de Von Neumann,
les langages de programation, different niveaux pour different besoins

Machine de turing



Architecture de von Neumann



Qu’est-ce qu’un programme ?

Les machines séquentielles, machine de Turing, architecture de Von Neumann,
les langages de programation, different niveaux pour different besoins

Tout commence par l’assembleur…

```
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
```

Qu’est-ce qu’un programme ?

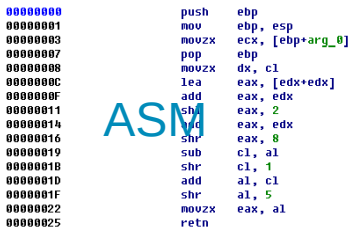
Les machines séquentielles, machine de Turing, architecture de Von Neumann,
les langages de programation, different niveaux pour different besoins

Assembly

C

C++

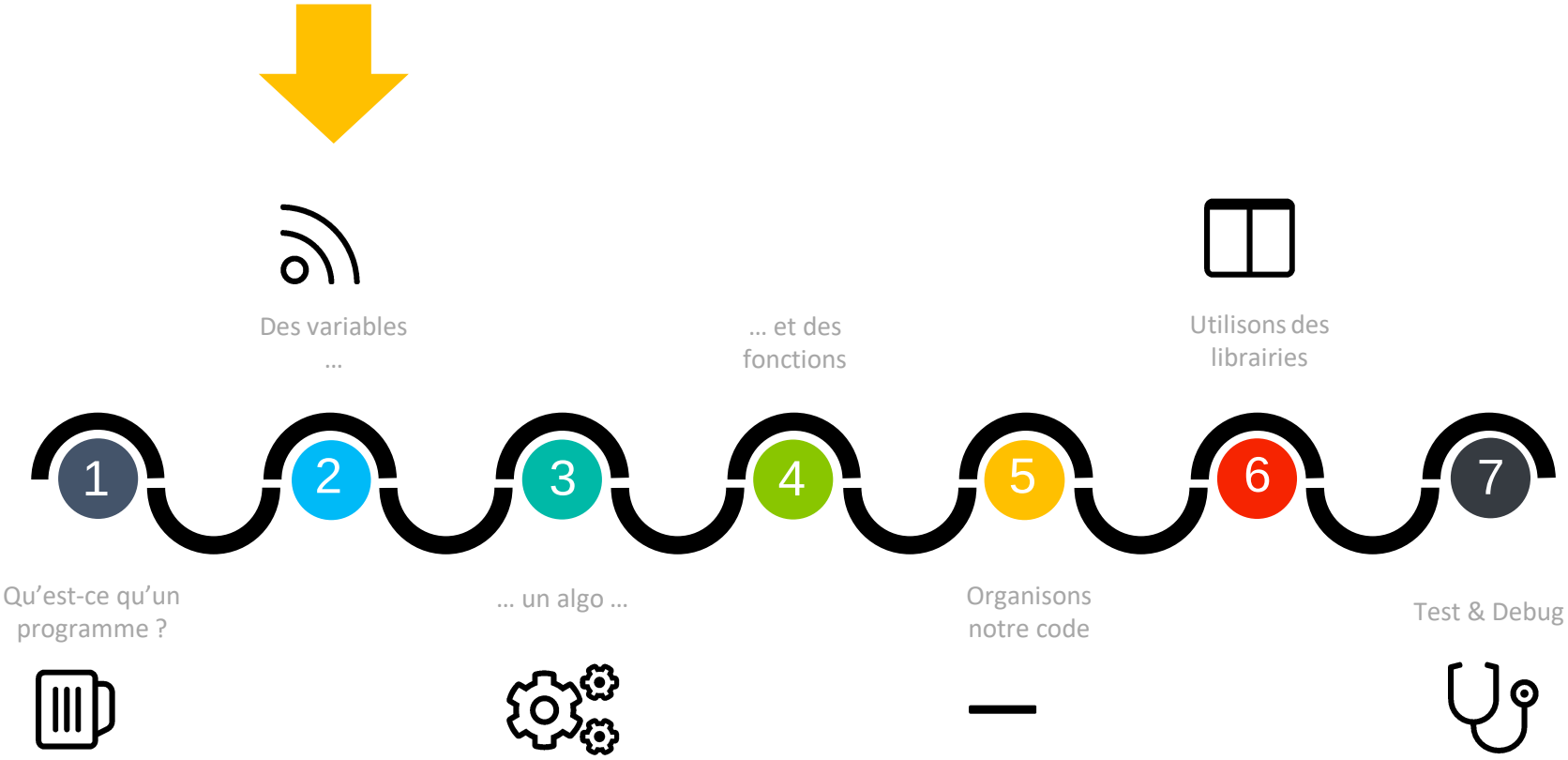
Python



Et python parmi les langages ?

Caractéristiques, cas d'utilisation, entreprises qui l'utilisent

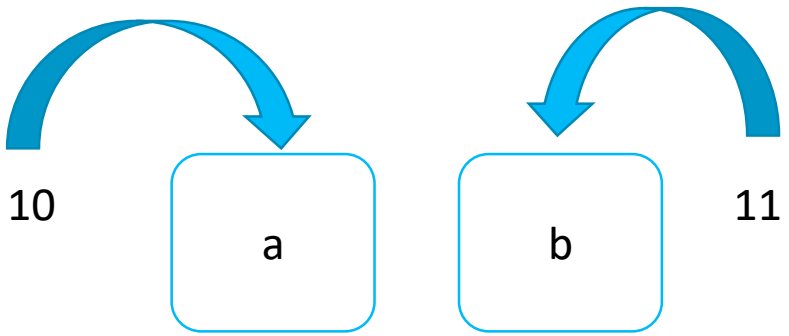
Des variables



Les variables

Les variables, convention de nommage, les types (parenthèse typage dynamique), la composition (types hétérogènes)

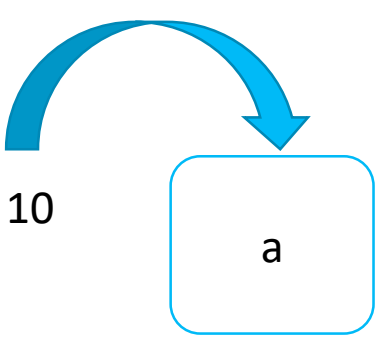
```
a = 10 # 10 → a
b = 11 # 11 → b
print(a) # affiche le contenu de la variable a
print("a") # affiche la chaine de caractère "a"
print(b) # affiche le contenu de la variable b
print(a + b) # affiche le résultat de l'opération
print("a + b") # affiche la chaine de caractère "a+b"
```



Les variables

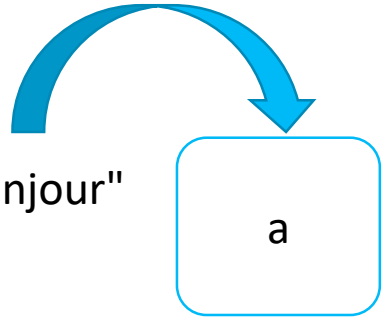
Les variables, convention de nommage, les types (parenthèse typage dynamique), la composition (types hétérogènes)

```
a = 10 # 10 → a
print(type(a)) # affiche le type de la variable a
a = "bonjour" # "bonjour"→ a
print(type(a)) # affiche le type de la variable a
a = True # True → a
print(type(a)) # affiche le type de la variable a
```



type(a) → int

Puis…



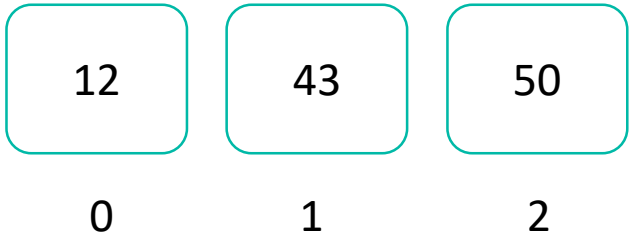
type(a) → str

Les variables

Les listes

Entrée [3]:

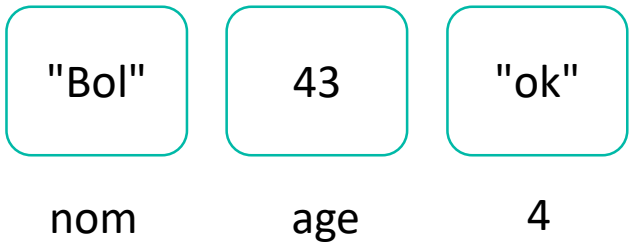
```
prix_des_articles_du_panier = [12,43, 50] # Crée une liste d'entiers
print(type(prix_des_articles_du_panier)) # Affiche le type de la liste

liste_quelconques = [12, True, "50", [1, 4, 0]] # Crée une liste quelconque
print(type(liste_quelconques)) # Affiche le type de la liste

print(prix_des_articles_du_panier[0]) # Affiche le premier élément de la liste
print(prix_des_articles_du_panier[2]) # Affiche le troisième élément de la liste
print(prix_des_articles_du_panier[-1]) # Affiche le dernier élément de la liste
```



Les variables

Les dictionnaires

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



Les variables

Les variables, convention de nommage, les types (parenthèse typage dynamique), la composition (types hétérogènes)

	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() permet d'avoir le hash d'un hashable
- Iterable => methode __iter__ qui renvoie une liste ou un générateur
- Hashable => methode __hash__ qui renvoie un hash unique
- Indexable =>methode __getitem__(self, index)
- sliceable =>methode __getitem__(self, slice)

Les variables

Les operations sur entiers, float et complexes

x + y	Addition
x - y	Soustraction
x * y	Multiplication
x / y	Division
x // y	Division entière
x % y	Reste
-x	Opposé
+x	
x ** y	Puissance

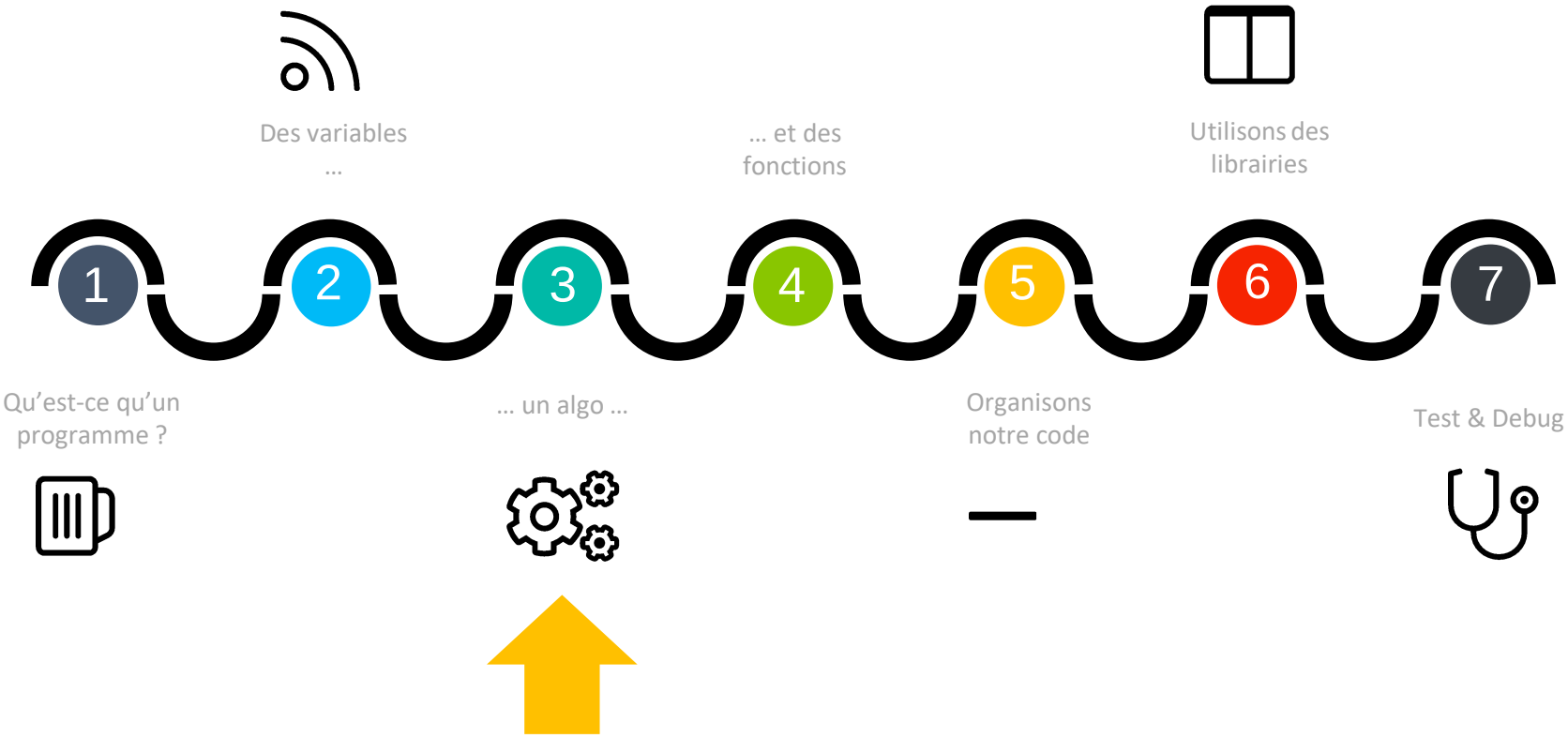
Les variables

Les operations sur les séquences

x not in s	False si s contient x, sinon True
s1 + s2	Concaténation
s * n	Répétition
s[i]	Élément à l'indice ou clef i
len(s)	Taille de la chaine
min(s)	Plus petit élément de la séquence
max(s)	Plus grand élément de la séquence
s.index(x)	Indice de la première occurence de x
s.count(x)	Nombre total d'occurrences de x

Un Algo…

Quelles pieces pour un algo ?



Un algo

La boucle WHILE

```
nb = 7
i = 0

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

Attention à l’indentation !

Un algo

La boucle FOR

```
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)
```

Un algo

Les conditions

```
name = 'Mickael'
if name == 'Mickael':
    print('Bonjour Mickael')
elif name == 'Laetitia':
    print('Bonjour Laetitia')
else:
    print('Vous n\'avez pas le droit de rentrer’)
```

Attention au double "=" !!

Un algo

Opérateurs de comparaison

<	Strictement inférieur à
>	Strictement supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

Un algo

Opérateurs de comparaison

- or
- not
- and

Un algo

Time for Fizz Buzz !!

Un autre exemple d’algo

Le tri bulle

Les inputs et templates

Interagir avec l'utilisateur

```
name = input('Quel est votre nom ? ')
age = int(input("quel est votre âge ? "))

"Ma variable : %type" % var
"Mes variables : %type, %type" % (var1, var2)
"Resultat : %(val)type %(unit)type" % {'val':var1, 'unit':var2}

type est d : entier - f : flottant - s : chaîne de caractère - c : caractère - o : octal - x : hexadécimal - c : caractère

Précision pour les float :
• "Resultat: %.2f" % 3.141592653589793
```

Recherche par dichotomie

Les fonctions

Definition vs Exécution

```
def dire_bonjour():
    print('Bonjour Monsieur!’)


dire_bonjour() #'Bonjour Monsieur!'
```

Les fonctions

Les paramètres (ordonnés, par default…)

```
def dire_bonjour():
    print('Bonjour Monsieur!')

def dire_bonjour(name):
    print('bonjour ' + name)

def dire_bonjour(name, name2='', name3='toto'):
    print('bonjour ' + name + ' ' + name2)
```

Les fonctions

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
```

Affichage d’un board

Librairies externes

Pip, parenthèse environnement, Selenium

Installation d’un module dans l’invite de commande

pip install selenium

Création d’un récapitulatif des modules installés

pip freeze > requirements.txt

Installation d’un groupe de modules

pip install –r requirements.txt

Création d’un environnement de développement

virtualenv monenv

Activation / Désactivation de l’environnement

monenv/Scripts/activate

deactivate

Exemple avec Selenium

Organiser son code

Commencent par :

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

Doit contenir un __init__.py :

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

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

Organiser son code

Packages & Modules

```
import MyModuleLibrary.MyModule
import MyModuleLibrary.MyModule2

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

Test & Debug

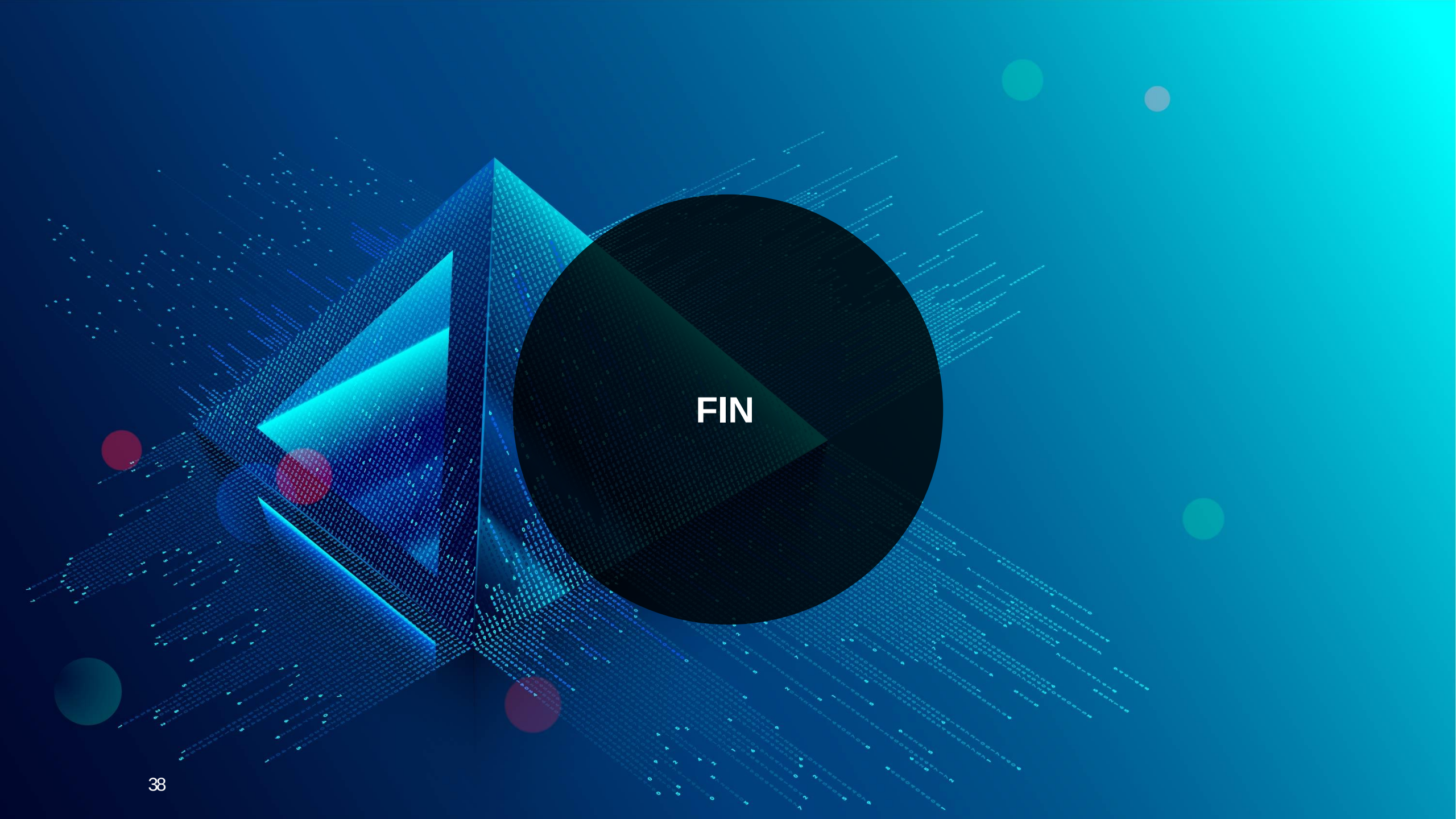
```
def add(a, b):
    """
        :Example:
        >>> add(1, 1)
        2
        >>> add(2.1, 3.4)
        5.5
    """
    return a + b


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) liste quelques lignes de code avant et après
- n : (next) exécute ligne suivante
- s : (step in) entre dans la fonction
- r : (return) sort de la fonction
- unt : (until) si dernière ligne boucle, reprend jusqu'à l'exécution boucle
- q : (quit) quite brutalement le programme
- c : (continue) reprend l'exécution


FIN