FMS ACADEMY BY FMS

FACYLITIES MULTI SERVICES

Une entreprise ordinaire à l'agilité extraordinaire

FORMATION PYTHON AWS

INTRODUCTION À PYTHON

martial.bret@fms-ea.com

06 49 71 51 16

Version : 1.0

DMAJ : 13/10/23

Module : DEV-PYTH-001
Python

01 – Connaitre l'historique et les atouts du langage Python
02 – Ecrire un programme Python et l'exécuter en ligne de commande
03 – Utiliser PyCharm pour écrire un programme Python
04 – Manipuler les types scalaires
05 – Manipuler les types conteneurs
06 – Utiliser les structures de contrôle de flux
07 – Ecrire des fonctions + portée des variables/paramètres
08 – Gérer les exceptions

Sommaire

- Historique du langage
- Caractéristiques principales
- Forces et faiblesses de Python
- Chiffres clés de sa communauté
- Installation Python et IDEs
- Premier programme Python
- Prise en main de PyCharm
- Aperçu de Thonny et de la gestion des paquets
- Les types scalaires et leurs opérateurs
- Les types conteneurs : séquentiels, ensembles, dictionnaire
- Contrôle de flux : choix, itération
- Fonctions, portée des variables et paramètres
- Programmation fonctionnelle
- Exceptions
- Ressources

Historique du langage

- 1991 : création par Guido van Rossum
  - travaille aux Pays-Bas au CWI sur projet Amoeba (système d'exploitation distribué)
  - conçoit Python à partir du langage ABC, version 0.9.0 publiée sur un forum Usenet.
- 2001 :
  - Python 2.1 (← Python 1.6.1 et Python 2.0)
  - naissance de la PSF (Python Software Fundation)
- 2008 : Python 3.0
  - répare certains défauts de Python 2 + nettoyage bibliothèque standard
  - casse compatibilité descendante/ascendante avec Python 2
- 2009 – 2023 : Python 3.1 – 3.12
- Plus de détails : https://fr.wikipedia.org/wiki/Python_(langage) > Historique

Caractéristiques 1 / 2

- Portable sur de très nombreux *OS*, libre, gratuit
- Interprété (et portable sur de nombreux *OS*)
  - fichier à l'extension `.py` exécutable directement
  - mais *bytecode* intermédiaire produit (parfois stockés en fichier `.pyc`), puis exécuté (machine virtuelle Python)
- Lisible, visuellement épuré
  - peu de constructions syntaxiques
  - blocs identifiés par indentation
- Simple
  - gestion automatique de la mémoire *(garbage collector)*
  - typage dynamique fort
  - entiers en précision « infinie »

```
def factorielle(n):
    if n < 2:
        return 1
    else:
        return n * factorielle(n - 1)
```

```
def factorielle(n):
    return n * factorielle(n - 1) if n > 1 else 1
```

```
factorielle(40)
815915283247897734345611269596115894272000000000
```

Caractéristiques 2/ 2

- Haut niveau
  - types de données évolués
    - liste, ensemble, dictionnaire, objet…
  - classes, fonctions, méthodes > objets dits de *première classe*
  - système d'exceptions
  - dynamique
  - réflexif > ex. : un objet peut
    - se rajouter ou s'enlever des attributs ou des méthodes
    - changer de classe en cours d'exécution
- Multiparadigme : procédural, objet, fonctionnel…
- Multithreadé (mais…)
- Bibliothèque standard riche, très nombreuses bibliothèques (paquets) disponibles dans tous les domaines
- Extensible (C, C++) | Extensions (Gimp, Google docs…)

```
f = factorielle
x = 4
print(f(x))
```

```
def calculer(fonction, nombre):
    return fonction(nombre)

print(calculer(factorielle, 4))
```

```
code = 'factorielle(4)'
print(eval(code))
```

Forces | Faiblesses 1/2

Aucun langage de programmation n'est parfait.

Il n'existe même pas un langage meilleur que d'autres.

Il n'y a que des langages en adéquation ou peu conseillés pour des buts particuliers. (Herbert Mayer)

Python n'est pas le meilleur langage pour quoi que ce soit.

Non. Python n'est pas le meilleur, mais seulement très bon.

Toutefois, il est très bon pour tout. (tutoriel « Python, de zéro »)

Forces | Faiblesses 2/2

+

- lisibilité et compacité, expressivité (1)
- évaluation paresseuse (2)
- nombreuses bibliothèques
  - IHM : Tk, wxPython, pyGTK, pyQt
  - Web : Flask, Django, FastAPI
  - science : Numpy, SciPy, Pandas, Matplotlib, Seaborn
  - *ML* : Scikit-learn , Tensorflow, Keras, Pytorch
  - OpenCV, Networkx, NLTK
- gestion automatique de la mémoire
- typage dynamique
- dynamicité et liberté induite

-

- (grande) lenteur
- modèle objet « impur », pas de niveaux de visibilité
- attributs spéciaux
  - `__name__` | `__doc__`
- méthodes spéciales
  - `__init__` | `__str__` `__eq__` | `__add__`
- pluralité des modalités de certains mécanismes

(1)

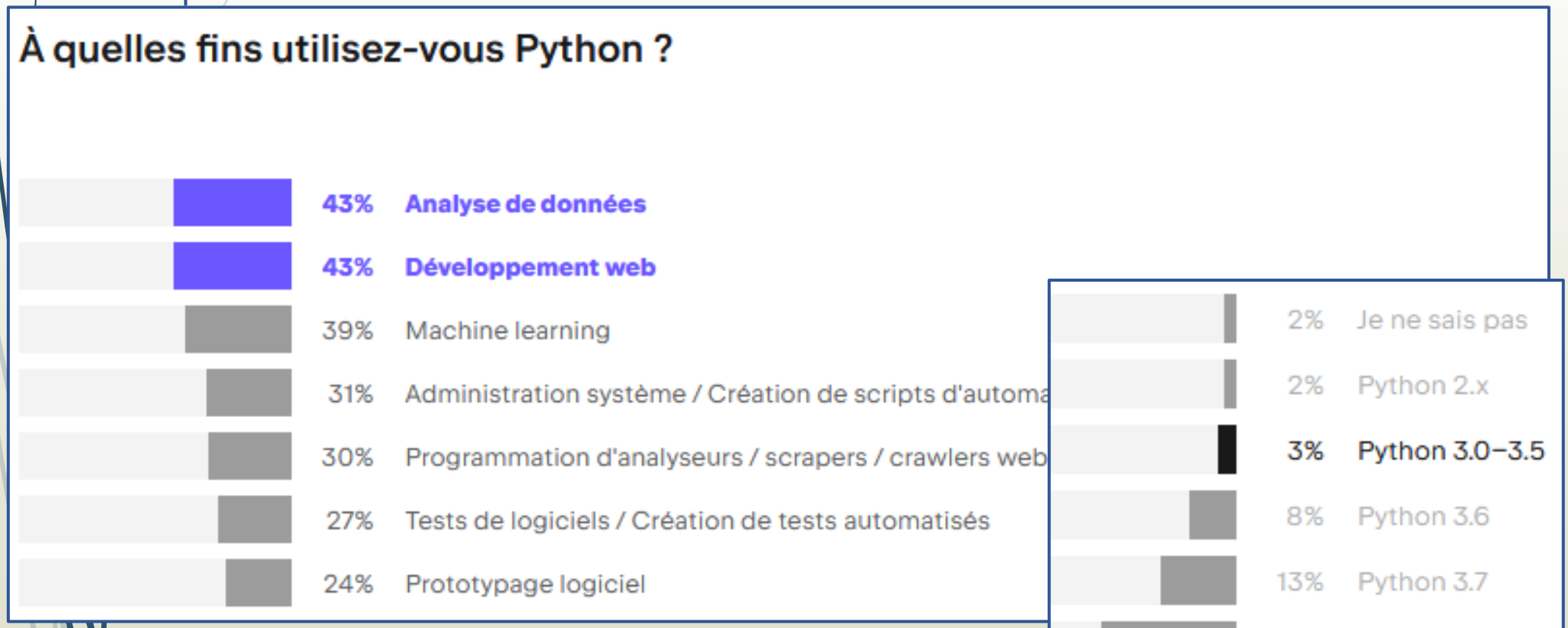
```
carres_nb_impairs = [n ** 2 for n in range(10) if n % 2 > 0]
[1, 9, 25, 49, 81]
```

(2)

```
gros_carres = (n ** 2 for n in itertools.count() if n ** 2 >= 1000)
next(gros_carres) → 1024
```

Chiffres clés de la communauté Python

Etude Jetbrains Python 2022 | Enquête de la Python Software Foundation et JetBrains



Quelles versions de Python utilisez-vous le plus ?

Installation Python et IDEs

- Python
  - https://www.python.org/downloads/windows/
    - Python 3.11.6 ou Python 3.12
    - *Windows installer (64-bit)*
    - *Optional Features* > tout cocher
  - compléments :
    - Alternative Python Implementations
    - PyScript
- IDEs + SonarLint
  - PyCharm > Community Edition
  - Visual Studio Code + extensions Python & Pylance
  - Thonny : simple, orienté apprentissage programmation
    - (détails d'installation plus loin)

Premier programme Python 1/2

- `Teams/1-cours/1-master_class/factorielle.py`
- Powershell

```
> python .\factorielle.py
Entrer un nombre entier positif : 6
La factorielle de 6 vaut 720.
```

- Git Bash

```
$ ls -l factorielle.py
-rwxr-xr-x 1 BretM 1049089 330 Oct  6 11:01 factorielle.py*

$ ./factorielle.py
Entrer un nombre entier positif : 6
La factorielle de 6 vaut 720.
```

- *Démarrer > Python > IDLE*
  - *File > Open*
  - *Run > Run Module*

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

def factorielle(n):
    if n < 2:
        return 1
    else:
        return n * factorielle(n - 1)

if __name__ == '__main__':
    nb = int(input("Entrer le nombre entier positif : "))
    fact_nb = factorielle(nb)
    print(f"La factorielle de {nb} vaut {fact_nb}.")
```

1 cf. Shebang

Premier programme Python 2/2

- *Teams*/1-cours/1-master_class/factorielle.py
- Powershell

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

```
> python
Python 3.11.5 …
Type "help", "copyright", "credits" or "license" for more information.
>>> import factorielle
>>> factorielle.factorielle(6)
720
>>> exit()
```

```
> python
Python 3.11.5 …
Type "help", "copyright", "credits" or "license" for more information.
>>> from factorielle import factorielle
>>> factorielle(6)
720
>>> ^Z
```

Prise en main PyCharm

- Fonctionnement similaire à Intellij IDEA (JetBrains)
- Reproduction premier programme : *File* > *New Project*…
  - *Location:* répertoire contenant `factorielle.py`
  - *Python Interpreter:* > *Previously configured interpreter* >
    - *Add Interpreter* > *Add Local Interpreter* > *System Interpreter* (la première fois)
    - *Interpreter:* Python 3.11
  - décocher *Create a main.py welcome script*
  - *Create* > *Create from Existing Sources*
- Volets spécifiques à Python
  - Python Console
  - Python Packages
- Débogueur

Aperçu Thonny, gestion des paquets

- https://github.com/thonny/thonny/wiki/Windows#installing-thonny-and-python-separately >

```
> pip list
Package    Version
---------- -------
pip        23.2.1
setuptools 65.5.0

> pip install thonny
Collecting thonny
  …
Installing collected packages: pyserial, …, thonny-4.1.3

> pip list
Package            Version
------------------ -------
astroid            3.0.0
…
thonny             4.1.3
…

> thonny
```

- cf. vidéo Introducing Thonny

Types scalaires, opérateurs 1/2

- entier

```
a = 2 + 3          # 5
b = a // 2         # 2
c = b * 4          # 8
d = c ** 2         # 64
e = d % 10         # 4
f = abs(e - 7)     # 3
type(f)  # <class 'int'>
g: int = 6
```

```
2023; 0b11111100111
0o3747; 0x7e7    #2023
~2023            # -2024
0b100 & 0b101    # 4
0b100 | 0b101    # 5
0b100 ^ 0b101    # 1
0b100 << 2       # 16
0b100 >> 2       # 1
```

- flottant

```
a = 5 / 2        # 2.5
b = 4 / 2        # 2.0
c = int(b)       # 2
d = float(3)     # 3.0
e = 2 * a + 1    # 6.0
type(e)  # <class 'float'>
f: float = e     # 6.0
g: int | float
```

- complexe – cf. module cmath

```
z1: complex = 2 + 2j
z2 = 5 - 2j
z3 = z1 - z2  # -3 + 4j
z3.real       # -3
z3.imag       # 4
abs(z3)       # 5.0
type(z3)  # <class 'complex'>
```

Types scalaires, opérateurs 2/2

- booléen

```
2 <= 4 < 7 < 8     # True
b1 = (2 == 2)      # True
b2 = (2 != 2)      # False
not b1             # False
b1 and b2          # False
b1 or b2           # True
int(b2), int(b1)   # 0, 1
b1 + b1            # 2
type(b1)  # <class 'bool'>
b: bool
```

- chaîne de caractères (suite)

```
voyelles = \
'a' 'e' 'i' 'o' 'u' 'y'  # 'aeiouy'
3 * 'hip ' + 'hourra’
# 'hip hip hip hourra'
voyelles.upper()         # 'AEIOUY'
voyelles.islower()       # True
type(voyelles)  # <class 'str'>
```

- chaîne de caractères

```
'j\'ai'
"j'ai"
'Il a dit "Oui"'
"Il a dit \"Oui\""
'Ligne 1\nLigne 2'
r'C:\repertoire\nom_fichier'
"""Ligne1
Ligne2"""
"""\
Ligne1
Ligne2
"""
```

```
str(2023)  # '2023'
bin(2023)  # '0b11111100111'
oct(2023)  # '0o3747'
hex(2023)  # '0x7e7'
```

cf. Type Séquence de Texte

Types conteneurs > séquentiels 1/2

- tuple [immuable]

```
t1 = (1, 2.0, 'hip', (2, 4))
t1[0], t1[1]  # (1, 2.0)
t1[-1]        # (2, 4)
t1[3][0]      # 2
len(t1)       # 4
2.0 in t1     # True
1 not in t1   # False

t2: tuple = 1, 2, 3, 4, 5
t2[1:3]       # (2, 3)
t2[-2:]       # (4, 5)
t2[0::2]      # (1, 3, 5)
t2[-1:1:-1]   # (5, 4, 3)
t2[:] == t2   # True
t2[:] is t2   # True
min(t2), max(t2)  # (1, 5)
t2.index(2)   # 1
tuple(n - 1 for n in t2)
# (0, 1, 2, 3, 4)
```

```
(a, b) = (4, 5)
a, b = b, a
(1, 2) * 3    # (1, 2, 1, 2, 1, 2)
((1, 2) * 3).count(1)  # 3
t3 = (3, )
(1, 2) + t3   # (1, 2, 3)
type(t3)  # <class 'tuple'>
tuple('abc')  # ('a', 'b', 'c')
```

- chaîne de caractères [immuable]

```
voyelles: str = 'aeiouy'
len(voyelles)      # 6
voyelles[0]        # 'a'
voyelles[5]        # 'y'
voyelles[1:3]      # 'ei'
'ou' in voyelles   # True
```

Types conteneurs > séquentiels 2/2

- liste [muable]

```
carres: list = [1, 3, 9, 16]
carres[1] = 4
carres.append(25)   # [1, 4, 9, 16, 25]
carres.extend([36, 49])
carres.pop()  # 49 => [1, 4, 9, 16, 25, 36]
carres[:] is carres  # False
[math.sqrt(n) for n in carres if n > 1]
# [2.0, 3.0, 4.0, 5.0, 6.0]
lettres = ['p', 'y', 'r', 's', 'o', 'n']
lettres[2:4] = ['t', 'h']
'-'.join(lettres)    # 'p-y-t-h-o-n'
lettres.remove('y')
lettres.sort()
''.join(lettres)     # 'hnopt'
'je suis là'.split()
# ['je', 'suis', 'là']
matrice = [[1, 0, 0], [0, 1, 0], [0,
type(matrice)        # <class list'>
```

En savoir plus :

- Type séquentiels
- fonctions natives >
  - enumerate()
  - sorted()
  - zip()

- range [immuable]

```
tuple(range(5))  # (0, 1, 2, 3, 4)
list(range(5))   # [0, 1, 2, 3, 4]
list(range(2, 5))  # [2, 3, 4]
3 in range(2, 5)   # True
list(range(2, 10, 2))   # [2, 4, 6, 8]
list(range(3, -3, -2))  # [3, 1, -1]
r = range(0, 20, 2)
r[-1]    # 18
r[:5]    # range(0, 10, 2)
type(r)  # <class 'range'>
```

Types conteneurs > ensembles

- set [muable], frozenset [immuable] cf. Types d'ensemble

```
voyelles = {'a', 'e', 'i', 'o', 'u', 'y'}
voyelles  # {'a', 'u', 'i', 'e', 'o', 'y'}
voyelles = frozenset('aeiouy')
voyelles  # frozenset({'a', 'i', 'e', 'o', 'u', 'y'})
len(voyelles)  # 6
{l.upper() for l in voyelles}
# {'O', 'U', 'Y', 'I', 'E', 'A'}
lettres: set = set('pommier')
lettres   # {'p', 'i', 'r', 'e', 'o', 'm'}
'p' in lettres  # True
voyelles - lettres  # {'a', 'u', 'y'}
lettres - voyelles  # {'p', 'r', 'm'}
voyelles ^ lettres  # {'p', 'a', 'r', 'm', 'u', 'y'}
voyelles | lettres  # {'p', 'a', 'i', 'r', 'e', 'o', 'm', 'u', 'y'}
voyelles & lettres  # {'e', 'o', 'i'}
lettres.add('s')
lettres   # {'p', 'i', 'r', 'e', 'o', 'm', 's'}
lettres.update(('a', 'e'))
lettres   # {'p', 'a', 'i', 'r', 'e', 'o', 'm', 's'}
type(voyelles)      # <class 'frozenset'>
type(lettres)       # <class 'set'>
```

Types conteneurs > dictionnaire

- dict [muable]

cf. Les types de correspondances

```
nb_du_mot: dict = {'un': 1, 'deux': 2}
nb_du_mot['deux']  # 2
nb_du_mot['trois'] = 3
nb_du_mot                # {'un': 1, 'deux': 2, 'trois': 3}
nb_du_mot = dict(un=1, deux=2, trois=3, quatre=4)
del nb_du_mot['quatre']
len(nb_du_mot)           # 3
nb_du_mot.keys()         # dict_keys(['un', 'deux', 'trois'])
nb_du_mot.values()       # dict_values([1, 2, 3])
'deux' in nb_du_mot      # True
nb_du_mot['quatre']      # lève l'exception KeyError
nb_du_mot.get('quatre', None)  # None
nb_du_mot.items()
# dict items([('un', 1), ('deux', 2), ('trois', 3)])
{mot: nb ** 2 for mot, nb in nb_du_mot.items()}
# {'un': 1, 'deux': 4, 'trois': 9}
nb_du_mot | {'quatre': 4}
# {'un': 1, 'deux': 2, 'trois': 3, 'quatre': 4}
type(nb_du_mot)          # <class 'dict'>
```

Contrôle de flux > choix

- `if ... elif ... else`

```
vitesse_max = 130
if (vitesse := 120) < vitesse_max:
    print(f"Votre vitesse de {vitesse} km/h est appropriée.")
    print("Pensez néanmoins à faire des pauses régulièrement.")
elif vitesse < vitesse_max * 1.1:
    print("Vous roulez trop vite, levez le pied.")
    vitesse -= 10
else:
    print("Vous roulez beaucou
    vitesse -= 20
```

- `match ... case`

```
# version simplifiée de la Belote où ♠ est toujours la couleur de l'atout
main_joueur = [('8', '♡'), ('9', '♠'), ('10', '♢'), ('V', '♣'), ('A', '♡')]
carte = random.choice(main_joueur)
match carte:
    case ('V', '♠') | ('9', '♠'):
        print("Vous avez un atout majeur !")
    case ('A', '♠') | ('10', '♠'):
        print("Vous avez un fort atout.")
    case (_, '♠'):
        print("Vous avez un atout.")
    case (hauteur, _) if hauteur == 10 or not hauteur.isnumeric():
        print("Vous avez une forte carte.")
    case _:
        print("Courage !")
```

Contrôle de flux > itération

- `while ... break ... continue ... else`

```
# quelques nombres de la suite de Fibonacci
# F0 = 0, F1 = 1, F{n} = F{n-1} + F{n-2}
a, b = 0, 1
while a < 100:
    print(a)
    a, b = b, a + b
else:
    print("...")
```

```
while True:
    pass
```

- `for ... break ... continue ... else`

```
# extrait de "La demeure entourée" (Jules Supervielle)
phrase = """Mais l'étoile se dit : « Je tremble au bout d'un fil,
Si nul ne pense à moi, je cesse d'exister. »"""
mots = re.split(r'\W+', phrase)
print("Cette phrase comporte :")
for longueur in range(1, 9):
    for mot in mots:
        if len(mot) == longueur:
            print("- au moins un", end='')
            break
    else:
        print("- aucun", end='')
    print(f" mot de {longueur} lettre", end='')
    print("s" if longueur > 1 else "")
```

Fonctions 1/2


```
def demande_entier(invite, nb_min=1, nb_max=None):
    """Demande à l'utilisateur de saisir un entier
    compris entre `nb_min` et (éventuellement) `nb_max`.
    :param invite: message d'invite à la saisie
    :param nb_min: nombre minimal
    :param nb_max: si pas None, nombre maximal
    :return: nb entier valide saisi par l'utilisateur
    """
    entier_valide = False
    # on suppose que c'est bien un nombre qui est entré
    entier = int(input(invite))
    if entier >= nb_min and \
        (nb_max is None or entier <= nb_max):
        entier_valide = True
    else:
        print("Merci d'entrer un entie

    if not entier_valide:
        entier = demande_entier(invite, nb_min, nb_max)

    return entier
```

1. Docstring
2. paramètre avec valeur par défaut
3. argument positionnel
4. argument nommé

```
nb_1 = demande_entier("Entrez un nombre entier positif : ")
nb_2 = demande_entier("Entrez un nombre positif ou nul : ", 0)
no_jour = demande_entier("Entrez un numéro de jour : ", 1, 31)
no_de = demande_entier("Entrez un numéro de dé : ", nb_max=6)
```

Fonctions 2/2

- Fonction imbriquée et fermeture

```
def fonction_trace(prefixe):
    def trace(message):
        print(prefixe + message)
    return trace

trace_avertissement = fonction_trace("Avertissement : ")
trace_erreur = fonction_trace("Erreur : ")

trace_avertissement("ça pourrait ne pas marcher")
trace_erreur("ça ne va pas marcher")
```

```
Avertissement : ça pourrait ne pas marcher
Erreur : ça ne va pas marcher
```

  - Portée des variables et paramètres
    - vers les fonctions englobées
    - `global` ou `nonlocal` nécessaire pour modifier une variable d'une portée englobante

- Fonction anonyme

```
def additionneur(n):
    return lambda x: x + n

plus_2 = additionneur(2)
plus_2(5), plus_2(12)  # 7, 14

plus_10 = additionneur(10)
plus_10(5), plus_10(12)  # 15, 22
```

Programmation fonctionnelle, aperçu

```
map(lambda x: 2 * x, range(1, 11))
# => 2, 4, 6, 8, 10, 12, 14, 16, 18, 20

map(lambda x, y: x + y, [1, 2, 3], [4, 5, 6])
# => 5, 7, 9

filter(lambda x: x % 2 == 0, range(10))
# => 0, 2, 4, 6, 8

sorted(['3', '1', '2', '11', '10'])
# ['1', '10', '11', '2', '3']

sorted(['3', '1', '2', '11', '10'], key=lambda n: int(n))
# ['1', '2', '3', '10', '11']

functools.reduce(lambda x, y: x + y, [1, 2, 3, 4])
# 10
functools.reduce(lambda x, y: x * y, [1, 2, 3, 4])
# 24
```

Exceptions, aperçu

```
def verifie_date(jour, mois, annee):
    def est_bissextile(annee):
        return annee % 4 == 0 and (annee % 100 != 0 or annee % 400 == 0)
    if mois == 2:
        jour_max = 28 + (1 if est_bissextile(annee) else 0)
    else:
        est_mois_30_jours = \
            (mois <= 7 and mois % 2 == 0) or (mois > 7 and mois % 2 != 0)
        jour_max = 30 + (0 if est_mois_30_jours else 1)
    if not 1 <= mois <= 12 or not 1 <= jour <= jour_max:
        raise ValueError("Date incorrecte")

def saisie_date():
    date = input("Entrez une date au format jj/mm/aaaa : ")
    jour, mois, annee = map(int, date.split('/'))
    try:
        verifie_date(jour, mois, annee)
    except ValueError as e:
        print(f"Erreur : {e}")
    else:
        print("Votre date est valide")
    finally:
        print("A bientôt")
```

Ressources

- https://docs.python.org/fr/3/ >
  - https://docs.python.org/fr/3/tutorial/index.html
  - https://docs.python.org/fr/3/library/stdtypes.html
    - Séquences Binaires (pour info)
  - https://docs.python.org/fr/3/library/functions.html >
    - `print() | round() | chr(), ord() | min(), max(), sum() reversed() | id()`
  - https://docs.python.org/fr/3/using/windows.html
- PEP (Python Enhancement Proposals)
  - PEP 8 – Style Guide for Python Code, PEP 20 – The Zen of Python
  - PEP 257 – Docstring Conventions, PEP 287 – reStructuredText Docstring Format
- Documentation du code
  - https://www.dataquest.io/blog/documenting-in-python-with-docstrings/
  - https://realpython.com/documenting-python-code
- https://frederic-lang.developpez.com/tutoriels/python/python-de-zero/