esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Langage Python

UP Web

AU: 2025/2026

United Nations Educational, Scientific and Cultural Organization
uniTwin
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia

EUR-ACE®
Délivrée par la Commission des Titres d'Ingénieur

CONFERENCE DES GRANDES ÉCOLES

cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

Plan

Introduction à Python

Qu'est-ce que Python ?

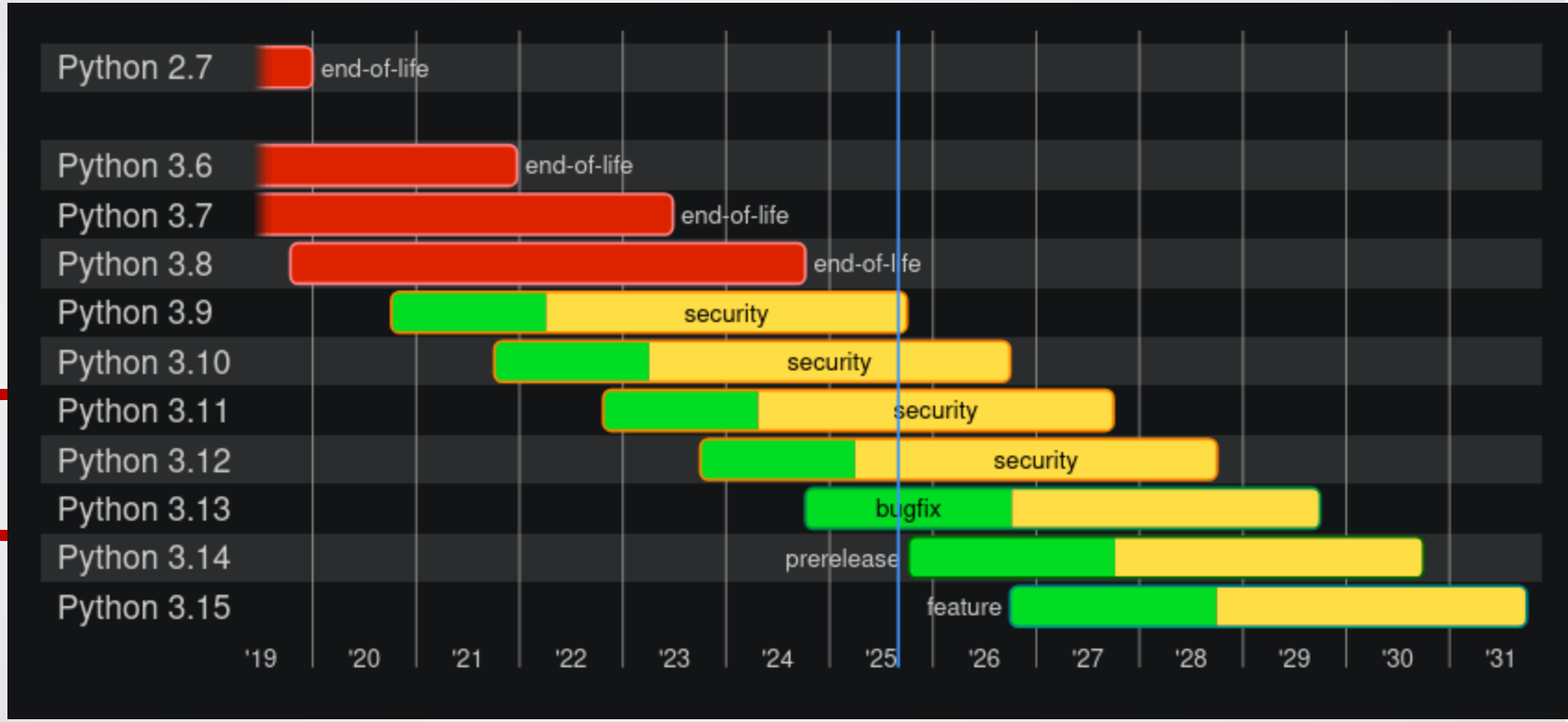
- Langage de programmation de ***haut niveau***, ***interprété*** et ***orienté objet***
- Syntaxe simple et lisible, mettant l'accent sur la productivité

Historique et philosophie du langage

- Créé par Guido van Rossum en 1991
- Philosophie "***Batteries Included***" : bibliothèques standard riches
- Principe "***There should be one-- and preferably only one --obvious way to do it***."

Introduction à Python

Versions acceptées



https://devguide.python.org/versions/#versions

Introduction à Python

Domaines d'application de Python

- Développement web (Django, Flask)
- Traitement de données et analyse (NumPy, Pandas, Matplotlib)
- Intelligence artificielle et apprentissage automatique (TensorFlow, PyTorch)
- Automatisation de tâches et scripts
- Développement de jeux et applications de bureau

Installation & Configuration

Téléchargement et installation de Python

- Rendez-vous sur le site officiel https://www.python.org/downloads/
- Choisissez la version appropriée pour votre système d'exploitation
- Suivez les instructions d'installation

Choix d'un environnement de développement (IDE)

- IDLE (Integrated Development and Learning Environment) : IDE intégré à Python
- PyCharm, Visual Studio Code, Spyder : IDE populaires avec de nombreuses fonctionnalités
- Jupyter Notebook : environnement interactif pour l'exploration et la présentation

Installation & Configuration

Exécution d'un programme Python

- Utilisation de l'interpréteur Python en ligne de commande
- Exécution de scripts Python avec l'extension .py
- Exemple de programme simple :

```
└──➤ python3
Python 3.13.7 (main, Aug 15 2025, 08:56:08) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Syntaxe de base

Commentaires

- Commentaires sur une seule ligne commençant par #
- Commentaires multiligne entre triple guillemets """ ou '''
- Exemple :

```
# This is a one line comment

"""
This is a multi-line comment
that spans several lines
"""
```

Variables et affectation

- Nommage des variables : lettres, chiffres, underscores
- Affectation avec le signe =
- Exemple :

```
age = 25
name = "Alice"
```

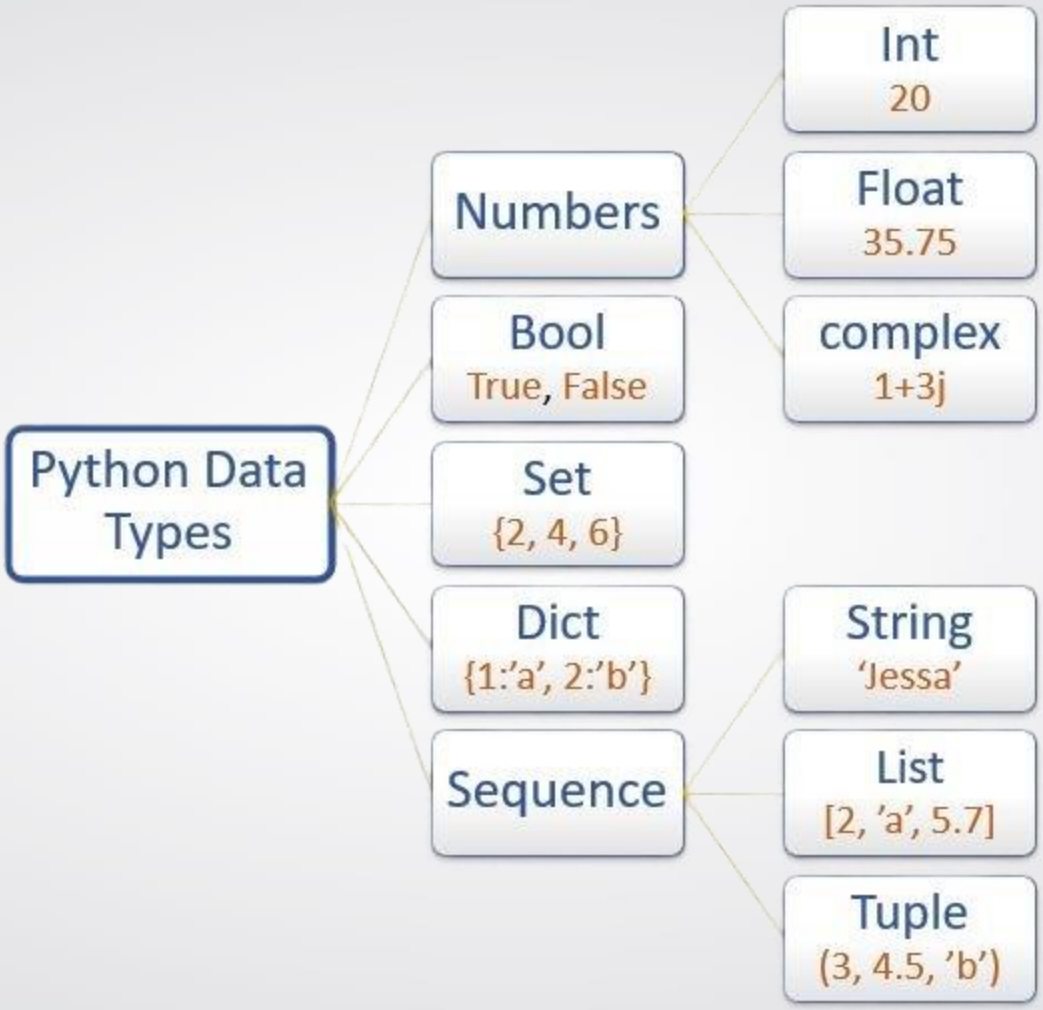
Syntaxe de base

Indentation et blocs de code

- Python utilise l'indentation pour délimiter les blocs de code
- Utilisez 4 espaces ou une tabulation pour indenter
- Exemple :

```
if 5 > 3:
    print("5 est plus grand que 3")
```

Types de données



Types de données

Types de Données Numériques

1. **Type int:** Représente les nombres entiers (positifs ou négatifs).

Exemple :

```
a = 10
b = -5
```

2. **Type float:** Représente les nombres à virgule flottante (décimaux).

Exemple :

```
pi = 3.14
temperature = -4.5
```

3. **Type complex:** Représente les nombres complexes.

Exemple :

```
z = 3 + 4j  # 3 est la partie réelle,
    4j est la partie imaginaire
```

Type Booléen

Type bool: Représente une valeur de vérité : True ou False.

Exemple :

```
est_majeur = True
est_enfant = False
```

Types de données

Types de Données de Séquence

1. **Type str (chaîne de caractères):** Représente une séquence de caractères.

Exemple :

```
nom = "Alice"
message = 'Bonjour, tout le monde!'
```

2. **Type list (liste):** Représente une collection ordonnée et modifiable d'éléments.

Exemple :

```
fruits = ["pomme", "banane", "cerise"]
```

3. **Type tuple (tuple):** Représente une collection ordonnée et immutable d'éléments.

Exemple :

```
coordinates = (10.0, 20.0)
```

Types de données

Types de Données Associatifs

1. **Type dict (dictionnaire):** Représente une collection non ordonnée de paires clé-valeur.

Exemple :

```
personne = {
    "nom": "Alice",
    "âge": 30,
    "ville": "Paris"
}
```

2. **Type set (ensemble):** Représente une collection non ordonnée d'éléments uniques.

Exemple :

```
nombres_uniques = {1, 2, 3, 4, 5}
```

Opérations sur les types de données

- Les opérateurs arithmétiques en Python sont:
  - + Pour l'addition
  - - Pour la soustraction
  - * Pour la multiplication
  - / Pour la division

```
>>> print(2+1)
3
```

```
>>> print(1+3+5*2)
14
>>> print((1+3+5)*2)
18
```

- L'usage des parenthèses est important!

```
x=2 ** 2
print(x)
```

```
4
```

- ** est utilisé pour la puissance

```
>>> print(11%2)
1
```

- % opérateur « modulo »

```
>>> print(17//4)
4
```

- // est l'opérateur « div »

Opérations sur les types de données

- Créer plusieurs variables en même temps

```
>>> x,y,z= 5,6,7
>>> print(x)
5
>>> print(y)
6
>>> print(z)
7
```

- Assigner la valeur d'une variable à une autre variable

```
>>> x=y=3
>>> print(y)
3
```

- Incrémentation
- Décrémentation
- Multiplication
- Division

```
>>> x+=1
>>> x-=1
>>> x*=0.95
>>> x/=2
```

Opérations sur les types de données

- Extraire le type d'une variable

```
>>> x=5
>>> print(type(x))
<class 'int'>
>>> x=5/2
>>> print(type(x))
<class 'float'>
>>> print(int(x))
2
```

- Conversion de type
- On perd de l'information lorsqu'on transforme des float en int (casting)

```
>>> resultat=5.5
>>> resultat_int=int(5.5)
>>> print(resultat_int)
5
>>> resultat_float=float(resultat_int)
>>> print(resultat_float)
5.0
```

- Vérification du type isinstance()

```
>>> x=5
>>> print(x)
5
>>> isinstance(x,int)
True
>>> isinstance(x,float)
False
```

Opérations sur les types de données

- Variables de type Booléen (bool)

```
>>> print(type(1))
<class 'int'>
>>> print(type(True))
<class 'bool'>
```

```
if 1 == True:
    print("1 et True sont équivalents!")
else:
    print("1 et True ne sont pas équivalents!")
```

```
1 et True sont équivalents!
```

- On obtient un résultat de type booléen lors d'une opération de comparaison

```
>>> 2 < 5
True
>>> 3 > 9
False
```

- Les opérateurs de comparaison

Inférieur strictement	<	**Supérieur**	>=
Supérieur strictement	>	**Egale**	==
Inférieur	<=	**Différent**	!=

Opérations sur les types de données

Opérateurs logiques

- L'opérateur **ET (and)** : Il est vrai seulement quand les deux opérateurs sont vrais, sinon il est faux

```
x= True and True
y=True and False
z=False and False
print(x)
print(y)
print(z)
```

```
True
False
False
```

- L'opérateur **OU (or)**: Il est vrai quand l'une des expressions est vraie, et faux si les deux opérateurs sont faux

```
x= True or True
y=True or False
z=False or False
print(x)
print(y)
print(z)
```

```
True
True
False
```

- L'opérateur **NON (not)**: Il évalue l'inverse de la valeur de vérité d'une expression, donc faux si l'expression est vraie, et vraie si l'expression est fausse

```
x= not True
y=not False
print(x)
print(y)
```

```
False
True
```

Les chaînes de caractères

- Le type **string** en python permet de manipuler les chaînes de caractère

```
x="je suis un String"
print(type(x))
```

- Accès à un string via son index

```
x="je suis un String"
print(x[3])
```

```
s
```

- L'opérateur + permet de concaténer deux chaînes de caractères

```
x= " je suis "
y= "un String"
print(x+y)
```

```
 je suis un String
```

- Multiplication des chaînes

```
x="abc"
print(x*3)
```

```
abcabcabc
```

- Obtenir une sous séquence de la chaîne

```
x="abcde"
print(x[0:3])
```

```
abc
```

```
x="abcde"
print(x[::2])
```

```
ace
```

- Vérifier l'existence d'une lettre dans une chaîne

```
x="abcde"
if 'a' in x:
    print("a est dans x")
```

```
a est dans x
```

Les chaînes de caractères

Longueur d'une chaîne de caractères

Une méthode prédéfinie **len** permet de retourner la longueur d'une chaîne de caractères.

```
>>> name_length = len("django")
>>> print(name_length)
6
```

Formatage des chaînes de caractères

Chaque type de donnée a son propre symbole

- %s pour les chaînes de caractères
- %c pour les caractères
- %d pour les entiers
- %f pour les floats

```
x="ceci est une %s de %s" % ("chaîne", "caractères")
print(x)
```

```
ceci est une chaîne de caractères
```

Les chaînes de caractères

Formatage des chaînes de caractères

Méthode str.format()

Introduite dans Python 2.7 et 3.0, cette méthode permet un formatage plus flexible.

```
name = "Alice"
age = 30
formatted_string = "Je m'appelle {} et j'ai {}
    ans.".format(name, age)
print(formatted_string)
# Affiche : Je m'appelle Alice et j'ai 30 ans.
formatted_string = "Je m'appelle {0} et j'ai {1}
    ans. {0} aime Python.".format(name, age)
print(formatted_string)
# Affiche : Je m'appelle Alice et j'ai 30 ans.
    Alice aime Python.
```

Formatage par f-strings (F-Strings)

Introduites dans Python 3.6, les f-strings permettent d'incorporer des expressions directement dans des chaînes de caractères.

```
name = "Alice"
age = 30
formatted_string = f"Je m'appelle {name} et j'ai
    {age} ans."
print(formatted_string)
# Affiche : Je m'appelle Alice et j'ai 30 ans.
formatted_string = f"Dans 5 ans, j'aurai {age +
    5} ans."
print(formatted_string)
# Affiche : Dans 5 ans, j'aurai 35 ans.
```

Les chaînes de caractères

- Mettre une chaîne en majuscule ou minuscule

```
message="bonjour les TWIN"
print(message.upper())
print(message.lower())
```

```
BONJOUR LES TWIN
bonjour les twin
```

- Formater des noms propres

```
message="bonjour ahmed ben salem"
print(message.title())
```

```
Bonjour Ahmed Ben Salem
```

- Calculer le nombre de caractères « x »

```
message="bonjour ahmed ben salem"
print(message.count("m"))
```

```
2
```

https://docs.python.org/3/library/stdtypes.html#string-methods

Application

1. Déclarer les variables suivantes
   - city = "Tunis"
   - temperature_max = 37
   - temperature_min = 12
   - unit_temp = "degrees Celsius"
   - notif = "Les prévisions pour aujourd\'hui pour" + city + ": Température entre " + str(temperature_min) + " et " + str(temperature_max) + " " + unit_temp+ ".
2. Simplifier cette chaîne de caractères en utilisant le formatage

Les listes (1/5)

- Une liste est une collection de données de différents type stockée de manière séquentielle, ce qui permet d'y accéder grâce à un index

```
Jours_semaine = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "samedi" , "dimanche"]
print(Jours_semaine[2])
debut_semaine=Jours_semaine[0:3]
print(debut_semaine)
```

```
Mercredi
['Lundi', 'Mardi', 'Mercredi']
```

```
ma_liste=[1,"ee",1.5,'abcde']
print(ma_liste)
```

```
[1, 'ee', 1.5, 'abcde']
```

- Stocker des structures de données dans une liste

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste]
print(ma_liste2)
```

```
['a', 10, [1, 'ee', 1.5, 'abcde']]
```

Les listes (2/5)

- Les éléments d'une liste sont indéxables

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste]
print(ma_liste2[0:2])
```

```
['a', 10]
```

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste]
print(ma_liste2[2::]) #du 2ème index jusqu'à la fin
print(ma_liste2[0:3:2]) #identifiant[debut:fin:interval]
```

```
[[1, 'ee', 1.5, 'abcde']]
['a', [1, 'ee', 1.5, 'abcde']]
```

- Remplacer un élément de ma liste

```
ma_liste=[1,"ee",1.5,'abcde']
print(ma_liste)
ma_liste[1]='a'
print(ma_liste)
```

```
[1, 'ee', 1.5, 'abcde']
[1, 'a', 1.5, 'abcde']
```

- On peut concaténer et multiplier des listes

```
ma_liste=[1,"ee",1.5,'abcde']
for element in ma_liste:
    print(element)
```

```
1
ee
1.5
abcde
```

Les listes (3/5)

- ***Split()*** : Transforme une chaîne de caractère (string) en liste

```
message= "bonjour mes chers !"
print(message)
message2=message.split()
print(message2)
type(message2)
len(message2)
message3= "bonjour-mes-chers !"
message4=message3.split()
print(message4)
message5=message3.split('-')
print(message5)
message6=" ".join(message5)
print(message6)
```

```
bonjour mes chers !
['bonjour', 'mes', 'chers', '!']
['bonjour-mes-chers', '!']
['bonjour', 'mes', 'chers !']
bonjour mes chers !
```

- ***Join()*****:** permet de retourner une chaîne de caractères contenant les éléments de la liste de chaînes ainsi que le caractère de jointure.

- ***Append()***: ajoute un élément à une liste**.**

```
liste_notes=["11","12.5","16","9","17"]
liste_notes2="-".join(liste_notes)
print(liste_notes)
print(liste_notes2)
liste_notes.append("19")
print(liste_notes)
```

```
['11', '12.5', '16', '9', '17']
11-12.5-16-9-17
['11', '12.5', '16', '9', '17', '19']
```

Les listes (4/5)

- Plusieurs fonctions nous permettent de manipuler les listes

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste]
ma_liste2.insert(2,"bonjour")
print(ma_liste2)
ma_liste2.remove("bonjour")
print(ma_liste2)
```

```
['a', 10, 'bonjour', [1, 'ee', 1.5, 'abcde']]
['a', 10, [1, 'ee', 1.5, 'abcde']]
```

```
>>> liste_notes = [11, 12.5, 16,9, 17]
>>> len(liste_notes)
5
>>> max(liste_notes)
17
>>> min(liste_notes)
9
>>> sorted(liste_notes)
[9, 11, 12.5, 16, 17]
>>> sorted(liste_notes, reverse= True)
[17, 16, 12.5, 11, 9]
```

```
>>> ma_liste=[1,2,3]
>>> ma_liste.append([1,2,3])
>>> print(ma_liste)
[1, 2, 3, [1, 2, 3]]
>>> len(ma_liste)
4
```

```
>>> ma_liste.count(1)#combien de fois un elt est présent dans une liste
1
>>> ma_liste_2=[2,3,4,5]
>>> ma_liste.extend(ma_liste_2)#ajouter une séquence d'elt
>>> print(ma_liste)
[1, 2, 3, [1, 2, 3], 2, 3, 4, 5]
>>> x=ma_liste.index(3)
>>> print("L'element 3 apparaît en premier à l'index %d" % x)
L'element 3 apparaît en premier à l'index 2
```

https://docs.python.org/3/tutorial/datastructures.html

Les listes (5/5)

- Parcourir les listes

```
ma_liste=[1,"ee",1.5,'abcde']
for element in ma_liste:
    print(element)
```

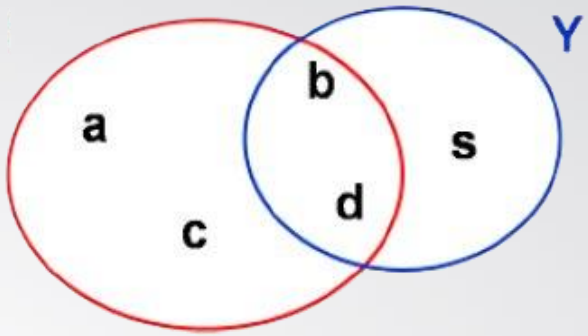
```
1
ee
1.5
abcde
```

```
jours_semaine = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "samedi" , "dimanche"]
for index,jour in enumerate(jours_semaine):
    print(index,jour)
```

```
0 Lundi
1 Mardi
2 Mercredi
3 Jeudi
4 Vendredi
5 samedi
6 dimanche
```

Les Collections "SET"

- Une autre structure de données en python qui permet d'avoir des éléments non-dupliqués est la structure « **set** ».
- On peut créer un set:
  - A partir d'une liste: my_set = set(ma_liste)
  - Vide: my_empty_set = set()
- On peut ajouter un élément dans un « set » par le mot clé **add**.




```
X, Y = set('abcd'), set('sbds')
print("X =", X) # X = {'a', 'c', 'b', 'd'}
print("Y =", Y) # Y = {'s', 'b', 'd'} : un seul élément 's'
print('c' in X) # True
print('a' in Y) # False
print(X - Y) # {'a', 'c'}
print(Y - X) # {'s'}
print(X | Y) # {'a', 'c', 'b', 'd', 's'}
print(X & Y) # {'b', 'd'}
```

Les dictionnaires (1/3)

- Les dictionnaires sont une structure de données qui sauvegardent une paire de données de la forme clé, valeur.
- Exemple:

```
elements = {'hydrogen': 1, 'helium': 2, 'carbon': 6}
```

- Pour accéder à un élément:

```
>>> print(elements['carbon'])
6
```

- Pour ajouter un nouvel élément:

```
>>> elements['lithium'] = 3
```

- Une autre manière pour accéder à un élément est l'usage de la méthode **get**

```
>>> print(elements.get('carbon'))
6
```

Les dictionnaires (2/3)

- ***Clear()***: permet de vider un dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> print(mon_dico)
{0: 'a', 1: 'b', 2: 'c'}
>>> print(type(mon_dico))
<class 'dict'>
>>> mon_dico.clear()
>>> print(mon_dico)
{}
>>> print(type(mon_dico))
<class 'dict'>
```

- ***Copy()***: permet de copier les éléments d'un dictionnaire dans un autre dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> mon_dico2=mon_dico.copy()
>>> print(mon_dico)
{0: 'a', 1: 'b', 2: 'c'}
>>> print(mon_dico2)
{0: 'a', 1: 'b', 2: 'c'}
>>> for x in mon_dico2:
        print(id(x))


1602082976
1602082992
1602083008
>>> for x in mon_dico:
        print(id(x))


1602082976
1602082992
1602083008
```

Les dictionnaires (3/3)

- ***Update()***: Merger deux dictionnaires

```
>>> mon_dico1={0:'a',1:'b',2:'c'}
>>> mon_dico2={0:'a',1:'b',2:'c',4:'z',5:'x'}
>>> mon_dico1.update(mon_dico2)
>>> print(mon_dico1)
{0: 'a', 1: 'b', 2: 'c', 4: 'z', 5: 'x'}
```

- ***Pop()***: Retirer un élément du dictionnaire grâce à sa clef et retourner sa valeur

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> print(mon_dico)
{0: 'a', 1: 'b', 2: 'c'}
>>> mon_dico.pop(2)
'c'
>>> print(mon_dico)
{0: 'a', 1: 'b'}
```

- ***Popitem()***: Retirer un élément du dictionnaire grâce à sa clef et retourner à la fois son indexe et sa valeur

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> mon_dico.popitem()
(2, 'c')
```

- ***Keys()***: Obtenir la liste des clefs du dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> x=mon_dico.keys()
>>> print(x)
dict_keys([0, 1, 2])
```

- ***Values()***: Obtenir la liste des valeurs du dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> x=mon_dico.values()
>>> print(x)
dict_values(['a', 'b', 'c'])
```

Les structures composites

- Selon notre besoin de représentation des données, il est possible de construire des structures composites de données.
- Exemple:

soit le dictionnaire suivant:

```
elements = {'hydrogen': {'number': 1, 'weight': 1.00794, 'symbol': 'H'},
            'helium': {'number': 2, 'weight': 4.002602, 'symbol': 'He'}}
```

- Il est possible d'accéder aux données ainsi:

```
>>> print(elements.get('hydrogen'))
{'number': 1, 'weight': 1.00794, 'symbol': 'H'}
>>> print(elements.get('hydrogen').get('weight'))
1.00794
>>> print(elements['hydrogen']['weight'])
1.00794
```

Les tuples

- Il est parfois utile de sauvegardé un ensemble corrélé de données.
- Exemple

  latitude_longitude = (10.3271, 11.5473)

```
>>> latitude_longitude[0]
10.3271
>>> latitude_longitude[1]
11.5473
>>>
```

- Un « tuple » peut avoir **n** dimensions.
- Une fonction peut retourner un tuple de valeurs.

Bilan des types

Type	Ordonné	Mutable	Parcours par indice	Parcours par éléments
str	✓	✗	✓	✓
list	✓	✓	✓	✓
set	✗	✓	✗	✓
dict	✗	✓	✗	✓
tuple	✓	✗	✓	✓

Les dictionnaires en Python sont ordonnés à partir de la version 3.7.
Cela signifie que l'ordre des éléments dans un dictionnaire est préservé dans cette version et les versions ultérieures.

Structures Conditionnelles

On utilise le mot clé **if** suivit par

- la condition
- Les instructions dans le bloc « if » sont précédées par une **tabulation**.

```
if poids_en_kg < 25:
    print("votre valise est acceptée")
```

Attention aux indentations dans vos scripts!

- On utilise les opérateurs booléens and, or et not pour composer les conditions

```
if number % 2 == 0:
    print("Le numéro " + str(number) + " est pair.")
else:
    print("Le numéro " + str(number) + " est impair.")
```

```
if poids_en_kg < 25:
    print("votre valise est acceptée")
elif poids >= 25 and poids < 40:
    print("Acceptée avec des frais")
else:
    print("refusée" )
```

Structures Itératives

Boucle "for"

- Utilisée pour itérer sur une séquence (comme une liste, un tuple ou une chaîne de caractères).

Syntaxe :

for élément in séquence:

 # bloc de code

Exemple de Boucle for :

```
fruits = ["pomme", "banane", "cerise"]
for fruit in fruits:
    print(fruit)

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

Structures Itératives

Boucle "while"

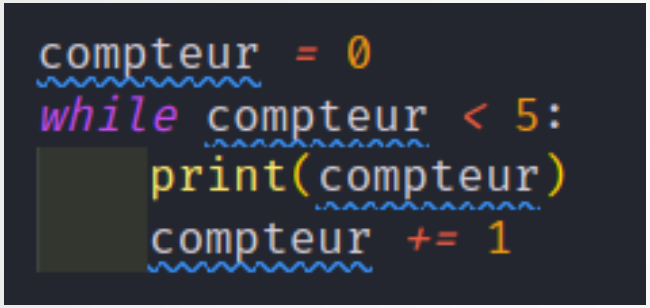
- Exécute un bloc de code tant qu'une condition est vraie.

Syntaxe :

while condition:

 # bloc de code

Exemple de Boucle while :



Structures Itératives

Instructions de Contrôle

1. **break**
   - Utilisée pour sortir d'une boucle prématurément.

Exemple :

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

2. **continue**
   - Utilisée pour sauter l'itération actuelle et passer à la suivante.

Exemple :

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Structures Itératives

Boucles Imbriquées

- Vous pouvez imbriquer des boucles pour parcourir des structures de données complexes.

```
for i in range(3):
    for j in range(2):
        print(f"i: {i}, j: {j}")
```

Les Fonctions

Paramètres d'une Fonction

Fonctions avec des Paramètres par Défaut

Vous pouvez définir des valeurs par défaut pour les paramètres.

Exemple

```
def saluer(nom="Invité"):
    return f"Bonjour, {nom}!"

print(saluer())          # Affiche : Bonjour, Invité!
print(saluer("Alice"))   # Affiche : Bonjour, Alice!
```

Fonctions avec un Nombre Variable d'Arguments

Vous pouvez utiliser *args pour passer un nombre variable d'arguments.

Exemple

```
def somme(*nombres):
    return sum(nombres)

print(somme(1, 2, 3))  # Affiche : 6
print(somme(5, 10, 15, 20))  # Affiche : 50
```

Les Fonctions

Définition d'une Fonction

Syntaxe

```
def nom_de_la_fonction(param1, param2):
    # bloc d'instructions
    return valeur
```

Exemple

```
def addition(a, b):
    return a + b

resultat = addition(5, 3)
print(resultat)  # Affiche : 8
```

Les Fonctions

Qu'est-ce qu'une Docstring ?

- Une **docstring** (documentation string) est une chaîne de caractères qui décrit le comportement d'une fonction.
- Elle doit être placée immédiatement après la définition de la fonction.
- Utilisez des triples guillemets (""") pour permettre plusieurs lignes.

Exemple

```python
def diviser(a, b):
    """Divise a par b.

    Args:
        a (float): Le numérateur.
        b (float): Le dénominateur.

    Returns:
        float: Le résultat de la division.

    Raises:
        ValueError: Si b est égal à zéro.
    """
    if b == 0:
        raise ValueError("Le dénominateur ne peut pas être zéro.")
    return a / b
```

Entrées et Sorties

Qu'est-ce que l'entrée/sortie (I/O) ?

- L'entrée fait référence aux données que le programme reçoit (par exemple, saisie de l'utilisateur).
- La sortie fait référence aux données que le programme renvoie (par exemple, affichage à l'écran).

Sortie en Python

- La Fonction ***print()***: Utilisée pour afficher des informations à l'écran.
- Syntaxe de base :

```
print(objet1, objet2, ..., sep=' ', end='\n')
```

Entrée en Python

- La Fonction input(): Utilisée pour obtenir des données de l'utilisateur.
- Syntaxe de base :

```
variable = input("message prompt")
```

Exceptions

Qu'est-ce qu'une exception?

- Une exception est un événement qui se produit pendant l'exécution d'un programme et qui interrompt le flux normal des instructions.
- Les exceptions peuvent être causées par des erreurs de programmation, des erreurs d'entrée/sortie, ou des conditions imprévues.

Exceptions intégrées

Python fournit plusieurs exceptions prédéfinies, telles que :

- ***ZeroDivisionError*** : Tentative de division par zéro.
- ***ValueError*** : Erreur de conversion de type.
- ***TypeError*** : Opération ou fonction appliquée à un objet de type incorrect.
- ***FileNotFoundError*** : Tentative d'accès à un fichier qui n'existe pas.

Syntaxe de base

```
try:
    # Code qui peut générer une exception
except NomDeLException:
    # Code à exécuter si l'exception se produit
```

Exceptions

Exemple

```
try:
    value = int(input("Entrez un nombre : "))
    result = 10 / value
except ValueError:
    print("Erreur : Ce n'est pas un nombre valide.")
except ZeroDivisionError:
    print("Erreur : Division par zéro.")
else:
    print("Le résultat est :", result)
```

POO: Les Classes

Classe

- Une classe est un modèle ou un plan qui définit les attributs et les méthodes des objets.

Syntaxe de base

class NomDeLaClasse*:*

 # Attributs et méthodes

Objet

- Un objet est une instance d'une classe. Il possède des attributs (données) et des méthodes (comportements).

```
class Chien:
    def aboyer(self):
        print("Woof!")

mon_chien = Chien()  # Création d'un objet
mon_chien.aboyer()   # Appel de la méthode
```

POO: Constructeur

__init__

- Le constructeur __init__ est une méthode spéciale qui est appelée lors de la création d'un objet.
- Il est utilisé pour initialiser les attributs de l'objet.

Exemple

```python
class Voiture:
    def __init__(self, marque, modele):
        self.marque = marque
        self.modele = modele

ma_voiture = Voiture("Toyota", "RAV4")
print(ma_voiture.marque, ma_voiture.modele)  # Affiche : Toyota RAV4
```

POO: Héritage

Héritage

- L'héritage permet de créer une nouvelle classe à partir d'une classe existante, en héritant ses attributs et méthodes.
- La nouvelle classe est appelée classe dérivée (ou enfant), et la classe d'origine est appelée classe de base (ou parent).

Exemple

```python
class Animal:
    def parler(self):
        print("L'animal fait un bruit.")

class Chien(Animal):
    def parler(self):
        print("Le chien aboie.")

mon_chien = Chien()
mon_chien.parler()  # Affiche : Le chien aboie.
```

POO: Variables Prédéfinies

Ci-après un résumé des principales variables prédéfinies pour les classes en python:

- **__name__** : Contient le nom de la classe.
- **__doc__** : Contient la docstring de la classe (si elle en a une).
- **__dict__** : Dictionnaire contenant les attributs de l'instance.
- **__bases__** : Tuple contenant les classes de base directes.
- **__class__** : Retourne la classe de l'objet.
- **__new__** : Méthode appelée pour créer une nouvelle instance de la classe.
- **__init__** : Méthode appelée pour initialiser une nouvelle instance de la classe.
- **__del__** : Méthode appelée quand l'instance est sur le point d'être détruite.
- **__str__** : Méthode appelée pour obtenir une représentation lisible de l'objet.
- **__dir__** : Méthode appelée pour obtenir une liste des attributs de l'objet.

Références

- https://docs.python.org/3/using/windows.html
- https://google.github.io/styleguide/pyguide.html
- https://docs.python.org/3/reference/datamodel.html#specialnames
- https://readthedocs.org
- https://doughellmann.com/blog/