Fiche Python – Fonctions essentielles en ITC

1 Généralités, affichage, saisie

Affectations avec =

Permet d'affecter la valeur à droite à la variable à gauche. Définit la variable à gauche.

```
a=5
b=1
c=a+b
```

affecte la valeur 6 à `c`.

`print()`

Affiche du texte ou des variables.

```
nom = "Alice"
print(nom)
print("Bonjour", nom)
```

`input()`

Permet de demander une saisie utilisateur.

```
prenom = input("Quel est ton prénom ? ")
print("Bonjour", prenom)
```

Attention : `input()` renvoie une chaîne de caractères.

```
age = int(input("Quel âge as-tu ? "))
```

Opérations

```
x+y, x-y   # somme, différence
x*y; x/y   # produit, division
x**y   # puissance
x//y, x%y   # quotient, reste de la div. euclidienne (que pour les int)
a+=   # pareil que a=a+
a-=   # pareil que a=a-
```

2 Tests, booléens

Un booléen a une valeur `True` ou `False`. On peut faire des tests d'égalités, inférieurs, ...

```
a=5; b=3;
a==b;a!=b  #test d'égalité, de différence
a<=b; a>b;  # test inférieur, supérieur, strict, non strict
P= a>=b; Q= a<b;   # affectation d'un boolen
P and Q
P or Q
not(P)
```

3 Conversions de types

```
x = 3
y = 3.14
z = "bonjour"
t = [1, 2, 3]
u = True

print(type(x))
print(type(y))
print(type(z))
print(type(t))
print(type(u))
```

donne

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'bool'>
```

On peut utiliser les fonctions `int()`, `float()`, `str()`, `bool()`, pour s'assurer d'un type.

```
      bool(1)   #renvoie True
```

4 Chaînes de caractères

`len()`

```
texte = "python"
print(len(texte))
```

`lower()` et `upper()`

```
texte = "PyThOn"
print(texte.lower())
print(texte.upper())
```

`replace()`

```
texte = "bonjour"
print(texte.replace("jour", "soir"))
```

`split()`

```
phrase = "PCSI informatique python"
mots = phrase.split()
print(mots)
```

`find()`

```
texte = "informatique"
print(texte.find("mat"))
```

5 Listes

`append()`

pour ajouter un élément à une liste

```
L = [1, 2, 3]
L.append(4)
print(L)  # renvoie [1,2,3,4]
```

`pop()`

pour enlever un élément, en le renvoyant.

```
L = [10, 20, 30]
x = L.pop(1) # stocke 20 dans x et l'enlève de la liste
print(x) # renvoie 20
print(L) # renvoie [10, 30]
```

Remarque : `pop()` concerne le dernier élément de la liste.

`len()`

pour donner la taille de la liste.

```
L = [10, 20, 30]
x = len(L)
print(x) # renvoie 3
```

`copy()`

pour créer une copie superficielle de L

```
M=L.copy() # permet de modifier les listes, sans toucher à l'original.
```

Attention : Les listes sont mutables. L'instruction `M=L` ne crée pas de copie de `L`. Si on modifie `M`, alors on modifie `L`.

Aller chercher les éléments dans une liste

```
s[i] # élément en position i
s[i:j] # tranche des éléments d'indices i à j-1
s[i:j:k] # tranche des éléments d'indices i à j-1 par pas de k
```

Remarque : On peut choisir `k=-1`, pour lister en sens inverse.

Opérations dans une liste

`+` permet de concaténer deux listes (ou deux chaînes de caractères) et `*` permet de dupliquer une liste autant de fois que l'on veut.

```
      L=[1,2,3]; M=[4,5];
      L+M #renvoie [1,2,3,4,5]
      M*2; 2*M; #renvoie
```

Définitions de listes en compréhension

```
carres = [x**2 for x in range(6)]
print(carres) # renvoie [0,1,4,9,16,25]
```

Avec condition :

```
pairs = [x for x in range(10) if x % 2 == 0]
print(pairs) # renvoie [0,2,4,6,8]
```

Fonctions moins utilisées dans les sujets

`insert()`

pour insérer un élément

```
L = [1, 3, 4]
L.insert(1, 2)
print(L) # renvoie [1,2,3,4]
```

`remove()`

pour enlever un élément

```
L = [1, 2, 3]
L.remove(2)
print(L)
```

`reverse()`

pour renverser une liste

```
L = [1, 2, 3]
L.reverse()
print(L) # renvoie [3, 2, 1]
```

`sort()`

pour trier une liste.

```
L = [4, 1, 8, 2]
L.sort()
print(L) # renvoie [1, 2, 4, 8]
```

6 Dictionnaires

Les dictionnaires permettent d'associer une clé à une valeur.
Exemple : un nom → une note.

```
d = {"Alice": 15, "Bob": 18}
```

Dans un dictionnaire :
— les clés doivent être uniques ;
— les valeurs peuvent être de n'importe quel type ;
— l'accès est très rapide.

Création d'un dictionnaire

Syntaxe générale

```
d = {cle: valeur, ...}
```

Exemple

```
notes = {
    "Alice": 15,
    "Bob": 18,
    "Claire": 17
}
```

Accès à une valeur : `d[c]`

Permet d'accéder à la valeur associée à la clé `c`.

```
notes = {"Alice": 15, "Bob": 18}

print(notes["Alice"])
```

Résultat :

```
15
```

Attention : si la clé n'existe pas, Python produit une erreur.

```
print(notes["Daniel"])
```

Ajouter ou modifier une valeur : `d[c2] = v2`

Ajouter une nouvelle clé

```
notes = {"Alice": 15}

notes["Bob"] = 18

print(notes)
```

Résultat :

```
{'Alice': 15, 'Bob': 18}
```

Modifier une valeur existante

```
notes["Alice"] = 16

print(notes)
```

Tester la présence d'une clé : `k in d`

```
notes = {"Alice": 15, "Bob": 18}

print("Alice" in notes)
print("Claire" in notes)
```

Résultat :

```
True
False
```

Très utile avant un accès :

```
if "Alice" in notes:
    print(notes["Alice"])
```

Nombre de clés : `len(d)`

```
notes = {
    "Alice": 15,
    "Bob": 18,
    "Claire": 17
}

print(len(notes))
```

Résultat :

```
3
```

Copie d'un dictionnaire : `d.copy()`

Effectue une copie superficielle.

```
d = {"a": 1, "b": 2}

e = d.copy()

print(e)
```

Attention :

```
e["a"] = 100

print(d)
print(e)
```

Résultat :

```
{'a': 1, 'b': 2}
{'a': 100, 'b': 2}
```

Le dictionnaire initial n'est pas modifié.

Parcourir les clés : `d.keys()`

```
notes = {
    "Alice": 15,
    "Bob": 18,
    "Claire": 17
}

for cle in notes.keys():
    print(cle)
```

Résultat :

```
Alice
Bob
Claire
```

En pratique, on peut souvent écrire directement :

```
for cle in notes:
    print(cle)
```

Parcourir les couples clé-valeur : `d.items()`

Très utile pour récupérer simultanément les clés et les valeurs.

```
notes = {
    "Alice": 15,
    "Bob": 18,
    "Claire": 17
}

for nom, note in notes.items():
    print(nom, note)
```

Résultat :

```
Alice 15
Bob 18
Claire 17
```

Parcourir les valeurs : `d.values()`

```
notes = {
    "Alice": 15,
    "Bob": 18,
    "Claire": 17
}

for note in notes.values():
    print(note)
```

Exemple classique en PCSI

Compter le nombre d'occurrences des lettres d'un mot.

```
mot = "informatique"
occurrences = {}
for lettre in mot:
    if lettre in occurrences:
        occurrences[lettre] += 1
    else:
        occurrences[lettre] = 1
print(occurrences)
```

Résumé des opérations importantes

Syntaxe	Rôle
`d = {c:v, ...}`	Création d'un dictionnaire
`d[c]`	Accès à la valeur associée à la clé `c`
`d[c2] = v2`	Ajout ou modification
`k in d`	Test de présence d'une clé
`len(d)`	Nombre de clés
`e = d.copy()`	Copie superficielle
`d.keys()`	Parcours des clés
`d.values()`	Parcours des valeurs
`d.items()`	Parcours des couples clé-valeur

7 Boucles, conditionnement

Conditionnement

```
if condition1:
       bloc_instructions
elif condition2:
       bloc_instructions
else:
       bloc_instructions
```

Itération et boucle for

```
range(n)  #entiers de 0 à n-1
range(a,b) # entiers de a à b-1
range(a,b,h) # entiers de a à b-1 avec un pas de h
```

Par exemple,

```
for i in range(4,6):
    print(i)
```

renvoie `4 5` .

Le forme générique d'une boucle for est

```
for e in s:
       instructions
```

Le forme générique d'une boucle while est

```
while condition:
       instructions
```

Remarque : Utilisation de `break` possible pour arrêter une boucle.

8 Fonctions définies par l'utilisateur

`def`

La forme générale d'une fonction

```
def f(x,y,...):
       bloc_instructions
       return z,...
```

Elle peut renvoyer zéro, une ou plusieurs valeurs

```
def carre(x):
    return x ** 2
```

```
def moyenne(a, b):
    return (a + b) / 2
```

```
def empiler(L, x):
    L.append(x)
```

9 Fichiers

```
f=open(fichier,'r')  # ouverture d'un fichier en lecture
f=open(fichier,'w')  # ouverture en écriture (remplacement)
f=open(fichier,'a')  # ouverture en écriture (rajout à la fin)
f.read()     # tout le fichier en une chaîne
f.readlines()   #liste des lignes
s.split()    # chaîne s séparée (par les espaces) en une liste
s.split(t)    # chaîne s séparée par le séparateur t en une liste
f.write(str)    # écrire la chaîne str dans le fichier
f.close()     # fermer le fichier
'\n'      # saut de ligne
'\t'      # tabulation
str(num)     # convertit num en chaîne
```

Lecture :

```
fichier = open("texte.txt", "r")
contenu = fichier.read()
print(contenu)
fichier.close()
```

Écriture :

```
fichier = open("texte.txt", "w")
fichier.write("Bonjour")
fichier.close()
```

Bonne pratique :

```
with open("texte.txt", "r") as fichier:
    contenu = fichier.read()
    print(contenu)
```

10 Module `math` (sera rappelé dans un sujet)

```
import math
```

`math.sqrt()`

```
print(math.sqrt(25))
```

`math.sin()`

```
print(math.sin(math.pi / 2))
```

`math.exp()`

```
print(math.exp(1))
```

`math.log()`

```
print(math.log(math.e))
```

11 Module `random` (sera rappelé dans un sujet)

```
import random
```

`random.randint()`

```
print(random.randint(1, 6))
```

`random.random()`

```
print(random.random())
```

`random.choice()`

```
couleurs = ["rouge", "bleu", "vert"]
print(random.choice(couleurs))
```

12 Fonctions mathématiques (sera rappelé dans un sujet)

`abs()`

```
print(abs(-7))
```

`round()`

```
print(round(3.14159, 2))
```

`pow()`

```
print(pow(2, 5))
```

`min()` et `max()`

```
print(min(4, 8, 1))
print(max(4, 8, 1))
```

`sum()`

```
notes = [12, 15, 18]
print(sum(notes))
```