

Types de valeurs

Entier (négatifs aussi)	18, 0o22, 0x12
Chaîne de caractères	" Hello World"
Réel	3.1415924
Booléen	True / False

Expressions + ordre des priorités

**	Exposant	←
-(unaire)	Négatif unaire	→
* / // %	Multiplication, division	→
+ -(binaire)	Négatif binaire	→
< <= > >= != ==	Comparaisons	→
not	NOT logique	→
and	AND logique	→
or	OR logique	→

Casting

int(whatever)	Transforme la variable en un int si possible
float(whatever)	Transforme la variable en un float si possible
str(whatever)	Transforme la variable en un string si possible

Alternatives

```
if value > 0 :
    #do stuff
elif value < 0 :
    #do other shit
else :
    #do the rest idk
```

Boucle while

```
i = 0
while i <= 4 : #on répète 5 fois
    print(i)
    i += 1
```

Faire attention à compter à partir de 0 jusqu'au nombre qu'on a besoin !!

Procédure vs fonction

Procédure	Une suite d'instructions
Fonction	Une suite d'instructions avec un retour d'expression

Fonction

```
def fonction(variable) :
    #do stuff
    return result
```

Une procédure n'a tout simplement pas de return. Sinon, c'est la même chose.

Documentation

Les commentaires peuvent être utilisés lors de la planification, de la description algorithmique et de l'étiquetage d'un code

Les docstrings décrivent les fonctions d'une fonction

Les commentaires ne peuvent être de plus de 72 caractères

Typehinting

```
def fonction(var: type) -> type:
```

Le type peut être str, int, float, tuple, list, dict, None ou bool

Boucle for

```
lst = ["weed", 1, 2, 3, 4]
for l in lst :
    print(l) # l = élément de la liste
```

le "for" prend chaque élément dans l'ordre en mémoire. Cela marche aussi pour les paires clé/valeur d'un dictionnaire et pour les éléments d'un tuple

Types de séquence

Strings	" Hello World"
Listes	[3, 7, " weed", (42,69)]
Tuples	(3, 7, " weed", [42,69])
Dictionnaires	{'Python': 'un langage progra', 'le tout':

Liste vs Tuple

Ressemblances	C'est tous les 2 des collections de tous types de données
Différences	Les listes sont muables par rapport aux tuples

Slicing

Input	lst = [0,1,2,3,4] second_lst = lst[0:5:2]
Output	second_lst = [0, 2, 4]

1^{er} opérateur : numéro du début de slicing
 2^{ème} opérateur : numéro de fin de slicing (!!! la position n'est pas incluse)
 3^{ème} opérateur : numéro de pas (la fin inclus mais pas le début) (comme on compte d'habitude)

Opérations sur les séquences

val in seq	Si val est dans la séquence seq
val not in seq	L'inverse d'au dessus
t = val[s:i:j]	Slicing (voir slicing)
len(seq)	Longueur de seq
min(seq)	La valeur minimale de seq

Opérations sur les séquences (cont)

max(seq)	La valeur maximale de seq
seq.index(val[, start[, end]])	Position de la première valeur val avec début et fin non obligatoires
seq.count(val)	Compte combien de fois on a val dans seq

Opérations sur les strings

str.lower()	Renvoie une copie de str en minuscule
str.upper()	Renvoie une copie de str en majuscule
str.replace(old, new)	Remplace la séquence dans le string old par new
str.split(delim)	Divise str à l'aide du délimiteur delim
str.strip()	Efface les espaces de str

Opérations séquences muables (listes)

val[s:i:j] = t	Slicing (!! Cela remplace val)
del val[s:i:j]	Efface en slicing
seq.append(val)	met val à la fin de seq
seq.clear()	Vide seq
seq += t	Étend seq avec t (si t est un string, il prend chaque caractère séparément)
seq *= rep	Répète seq et le remet dans la variable
seq.insert(before, val)	Insère la valeur après before
seq.pop(pos)	Affiche et efface dans la séquence (on compte à partir de 1 !!)
seq.remove(val)	Efface la 1 ^{ère} valeur dans seq
seq.reverse()	Inverse la séquence



Insérer un élément dans une séquence muable

Liste `list.append(truc)` ou `list.insert(pos, truc)`

Dictionnaire Refaire un dict avec la position ou `dict["key"] = value` pour le mettre en dernière position

Operations sur les dictionnaires

`dict.items()` Retourne les paires clé/valeurs dans une liste de tuples

`dict.keys()` Retourne les clés dans une liste

`dict.values()` Retourne les valeurs dans une liste

Dictionnaire dans un dictionnaire

`dict[key_1][key_2][key_n]`

Déballage séquence

C'est l'utilisation des tuples comme expression de gauche.

Cela permet de prendre plusieurs variables en même temps ou affecter des variables sans savoir exactement combien il y en a.

Principes de Clean Code

Préférer l'utilisation des `val` et `not val` dans les alternatives et les boucles

Utiliser des boucles `while` quand on a pas besoin de passer par toute la séquence, sinon utiliser une boucle `for`

Utiliser les fonctions le plus possible (mais sans exagérer non plus)

Clean Code (À NE PAS FAIRE !!)

```
if boolean == True and another_bool == False :
    #do stuff
else :
    #do something else
```

Fonctions utiles

`range(start, end, step)` Génère une séquence de nombres respectant un motif strict. `start` n'est pas obligatoire et non plus le `step`. Le `end` est

`print(whatever)` Imprime n'importe quoi sur la console (bon pour les debugs)

`type(whatever)` Nous dit le type de variable

`id(whatever)` Nous montre l'adresse où se situe la valeur de la variable

`random()` Nous donne un nombre au hasard entre 0 et 1

`random.randint(start, end)` Nous donne un nombre au hasard entre `start` et `end`. `start` et `end` sont obligatoires

`ord('a')` Transforme un caractère en son code Unicode décimal (`a = 97`)

`chr(97)` Transforme un code Unicode décimal en un caractère Unicode

