Notions de base

Calculer avec Python

- Sous Python 3 :
  - L'opérateur de division **/** effectue une division réelle
  - Pour obtenir une division entière (le résultat est un entier), utiliser **//**

❖ **Tester**

⇒ **20.5/3 vs 20.5//3**

⇒ **8.7/5**

Attention : taper le **.** et pas le **,**

Opérateurs

- Opérations mathématiques

  *, +, /, //, ** (exponentiation)

- Opérateur modulo : %

Fournit le reste de la division entière

❖ Testez :

10 % 3

10 % 5

=> Pour tester si un nombre a est divisible par un nombre b, il suffira de vérifier que (a % b) donne un résultat égal à 0.

Noms de variables

- Une séquence de lettres et de chiffres commençant par une lettre
- Seul les lettres ordinaires sont autorisées et le caractère _ (souligné)
- La casse est significative
- Pas utiliser les mots réservés

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

Affectations

- Le signe =

  n=7; m="Bonjour" ; pi=3.14159

⇒Les trois noms de variables sont des références mémorisées dans une zone de mémoire appelée espace de noms

⇒Les valeurs correspondantes sont situées ailleurs.

- Afficher la valeur

>>> n

>>> print(n)

❖Pour voir la différence, testez avec la variable m

Attention : ne pas oublier les parenthèses avec print

Affectations multiples

- Assigner une valeur à plusieurs variables simultanément

```
>>> x = y = 7
>>> x
7
>>> y
7
```

- Affectations parallèles

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Contrôle du flux d'exécution

- La description structurée d'une série d'action et de l'ordre dans lequel il convient de les effectuer s'appelle un **algorithme**
- Le « chemin » suivi par Python à travers un programme est appelé un **flux d'exécution**, et les constructions qui le modifient sont appelées les **instructions de contrôle de flux**
- Les **structures de contrôle** sont les groupes d'instructions qui déterminent l'ordre dans lequel les actions sont effectuées

Instruction *if*

- *If* (si) permet de tester la validité de la condition.
- *else* (sinon) permet de programmer une exécution alternative
- *elif (else if=sinon si)*

```
>>> a = 0
>>> if a > 0 :
...     print("a est positif")
... elif a < 0 :
...     print("a est négatif")
... else:
...     print("a est nul")
...
```

Opérateurs de comparaison

```
x == y        # x est égal à y
x != y        # x est différent de y
x > y         # x est plus grand que y
x < y         # x est plus petit que y
x >= y        # x est plus grand que, ou égal à y
x <= y        # x est plus petit que, ou égal à y
```

Exemple

```
>>> a = 7
>>> if (a % 2 == 0):
...     print("a est pair")
...     print("parce que le reste de sa division par 2 est nul")
... else:
...     print("a est impair")
...
```

Instructions imbriquées

```
if embranchement == "vertébrés":                    # 1
    if classe == "mammifères":                      # 2
        if ordre == "carnivores":                   # 3
            if famille == "félins":                 # 4
                print("c'est peut-être un chat")    # 5
        print("c'est en tous cas un mammifère")     # 6
    elif classe == "oiseaux":                       # 7
        print("c'est peut-être un canari")          # 8
print("la classification des animaux est complexe") # 9
```

Ré-affectation

- Ré-affectation : remplacer l'ancienne valeur d'une variable par une nouvelle

```
>>> altitude = 320
>>> print(altitude)
320
>>> altitude = 375
>>> print(altitude)
375
```

- Symbole *égale* utilisé pour une affectation est différent du symbole *égalité*
  - L'égalité est commutative : En mathématique, les écritures : a=7 et 7=a sont équivalentes alors qu'une instruction 375=altitude est illégale
  - L'égalité est permanente alors que l'affectation peut être remplacée

Exercice

```
>>> a, b, c, d = 3, 4, 5, 7
```

Comment échanger les valeurs des variables **a** et **b** ?

On veut que **a** contienne la valeur **4** et **b** la valeur 3.

```
a,b,c,d=3,4,5,7
e=a
f=b
a=f
b=e
```

L'instruction *while*

- L'instruction while (tant que) amorce une instruction composée

❖Testez

```
>>> a = 0
>>> while (a < 7):          # (n'oubliez pas le double point !)
...     a = a + 1           # (n'oubliez pas l'indentation !)
...     print(a)
```

- Eviter les boucles sans fin !!!

```
>>> n = 3
>>> while n < 5:
...     print("hello !")
```

Construction d'une suite mathématique : Suite de Fibonacci

- Une suite de nombres dont chaque terme est égal à la somme des deux termes qui le précèdent

```
>>> a, b, c = 1, 1, 1
>>> while c < 11 :
...     print(b, end =" ")
...     a, b, c = b, a+b, c+1
```

Lorsque vous lancez l'exécution de ce programme, vous obtenez :

```
1 2 3 5 8 13 21 34 55 89
```

- L'argument **end=" "** signifie qu'on souhaite remplacer le saut de la ligne par une simple espace.

```
>>> a, b, c = 1, 1, 1
>>> while c < 11 :
...     print(b, end =" ")
...     a, b, c = b, a+b, c+1
```

Variables	a	b	c
Valeurs initiales	1	1	1
Valeurs prises successivement, au cours des itérations	1 2 3 5 ...	2 3 5 8 ...	2 3 4 5 ...
Expression de remplacement	b	a+b	c+1

L’instruction *for*

```
for lettre in "hello":
        print (lettre)
for i in range(6):          #générateur de séquences d’entiers
        print (i)
```

Un petit bonus

- i=i+1
- i+=1

Exercices à faire

- Faire un test pour savoir si a est compris dans l'intervalle allant de 2 à 8
- Afficher les 10 premiers termes de la table de multiplication par 7.
- Afficher les 10 premiers termes de la table de multiplication par 7, en signalant au passage (à l'aide d'une astérisque) ceux qui sont des multiples de 3.
  - Exemple : 7 14 21* 28 35 42* 49 ….
- Afficher une suite de 10 nombres dont chaque terme soit égale au triple du terme précédent