

Types de base

entier, flottant, booléen, chaîne

```
int 783 0 -192
float 9.23 0.0 -1.7e-6
bool True False
str "Un\nDeux"
```

retour à la ligne

☞ immutables

Identificateurs

pour noms de variables, fonctions, modules, classes...

a...zA...Z suivi de a...zA...Z_0...9

- accents possibles mais à éviter
- mots clés du langage interdits
- distinction casse min/MAJ

☉ a toto x7 y_max BigOne

☉ 8y and for

Variables & affectation

=

☞ affectation ⇔ association d'un nom à une valeur

- évaluation de la valeur de l'expression de droite
- affectation dans l'ordre avec les noms de gauche

x=1.2+8+sin(y)

a=b=c=0 affectation à la même valeur

y,z,r=9.2,-7.6,0 affectations multiples

a,b=b,a échange de valeurs

x+=3 incrémentation ⇔ x=x+3 et /=

x-=2 décrémentation ⇔ x=x-2 %

x*=5 multiplication ⇔ x=x*2 ...

Instruction boucle conditionnelle

bloc d'instructions exécuté tant que la condition est vraie

while condition logique:

☞ bloc d'instructions

s = 0 } initialisations avant la boucle

i = 1 } condition avec au moins une valeur variable (ici i)

while i <= 100:

s = s + i**2

i = i + 1

print("somme:", s)

☞ faire varier la variable de condition !

Algo : $i=100$

$s = \sum_{i=1}^{100} i^2$

Logique booléenne

Comparateurs: < > <= >= == !=

(résultats booléens) ≤ ≥ = ≠

a and b et logique les deux en même temps

a or b ou logique l'un ou l'autre ou les deux

☞ piège : and et or retournent la valeur de a ou de b (selon l'évaluation au plus court).

☞ s'assurer que a et b sont booléens.

not a non logique

True } constantes Vrai/Faux

False }

Blocs d'instructions

instruction parente:

☞ bloc d'instructions 1...

☞ instruction parente:

☞ bloc d'instructions 2...

☞ instruction suivante après bloc 1

☞ régler l'éditeur pour insérer 4 espaces à la place d'une tabulation d'indentation.

Imports modules/noms

module truc ⇔ fichier truc.py

from monmod import nom1, nom2 as fct

☞ accès direct aux noms, renommage avec as

import monmod

☞ accès via monmod.nom1 ...

☞ modules et packages cherchés dans le python path (cf. sys.path)

Instruction conditionnelle

un bloc d'instructions exécuté, uniquement si sa condition est vraie

if condition logique:

☞ bloc d'instructions

Combinable avec des sinon si, sinon si... et un seul sinon final. Seul le bloc de la première condition trouvée vraie est exécuté.

☞ avec une variable x:

if bool(x)==True: ⇔ if x:

if bool(x)==False: ⇔ if not x:

if age<=18: etat="Enfant"

elif age>65: etat="Retraité"

else: etat="Actif"

Définition de fonction

nom de la fonction (identificateur)

paramètres nommés

def fct(x,y,z):

☞ documentation

☞ bloc instructions, calcul de res, etc.

return res

☞ valeur résultat de l'appel, si rien à retourner return None

☞ les paramètres et toutes les variables de ce bloc n'existent que dans le bloc et pendant l'appel à la fonction (penser "boîte noire")

Appel de fonction

r = fct(3, "hey", a)

stockage/utilisation d'une valeur d'argument de la valeur de retour par paramètre

☞ c'est l'utilisation du nom de la fonction avec les parenthèses qui fait l'appel

Maths

☞ nombres flottants... valeurs approchées !

Opérateurs : + - * / // % **

Priorités (...)

☞ priorités usuelles

Affichage

print("v=", 3, "cm :", x, " ", y+4)

éléments à afficher : valeurs littérales, variables, expressions

Options de print:

- sep=" " séparateur d'éléments, défaut espace
- end="\n" fin d'affichage, défaut fin de ligne

Saisie

s = input("Directives:")

☞ input retourne toujours une chaîne, la convertir vers le type désiré (cf. encadré Conversions).

Conversions

type (expression)

int("15") → 15

float("3.14") → 3.14

int(15.56) → 15

troncature de la partie décimale

Séquences d'entiers

range([début,] fin [,pas])

☞ début défaut 0, fin non compris dans la séquence, pas signé et défaut 1

range(5) → 0 1 2 3 4

range(2,12,3) → 2 5 8 11

range(3,8) → 3 4 5 6 7

range(20,5,-5) → 20 15 10

☞ range fournit une séquence immuable d'entiers construits au besoin

Indexation conteneurs séquences

pour les listes, chaînes de caractères...

index négatif	-5	-4	-3	-2	-1
index positif	0	1	2	3	4

lst=[10, 20, 30, 40, 50]

Accès individuel aux éléments par lst[index]

lst[0]→10

lst[1]→20

Nombre d'éléments

len(lst)→5

☞ index à partir de 0 (de 0 à 4 ici)

lst.append(val) ajout d'un élément à la fin

Sur les séquences modifiables (list), suppression avec del lst[3] et modification par affectation lst[4]=25

Instruction boucle itérative

bloc d'instructions exécuté pour chaque élément d'un conteneur ou d'un itérateur

for var in séquence:

☞ bloc d'instructions

Parcours des valeurs d'un conteneur

s = "Du texte" } initialisations avant la boucle

cpt = 0

variable de boucle, affectation gérée par l'instruction for

for c in s:

if c == "e":

cpt = cpt + 1

print("trouvé", cpt, "e")

Algo : comptage du nombre de e dans la chaîne.

☞ bonne habitude : ne pas modifier la variable de boucle