Algorithmique

et Python

Plan

Algorithme et Python

Définition

Un algorithme est une succession d'un nombre fini d'étapes réalisées dans un ordre précis et qui a pour but de résoudre un problème.

instructions dans un langage compréhensible par l'exécutant :

- humain $\Rightarrow$ langage courant.
- ordinateur $\Rightarrow$ langage de programmation (Python).
- algorithmique $\Rightarrow$ pseudo-code.

Python

Langage de programmation interprété.
Un programme en Python :

1. écrire texte contenant les instructions
2. demander à Python d'exécuter le programme.
3. Python lit le texte et suit les instructions.
4. Si tout va bien, il va au bout. Si il rencontre une erreur, il s'arrête et renvoi un message d'erreur.

Il faut une distribution Python et un éditeur de code pour l'utiliser. On va privilégier les tout-en-un.

- Sur ordinateur : Installer un logiciel (Spyder, Idle, Anaconda...)
- Sur internet, dans un navigateur :
  https://www.programiz.com/python-programming/online-compiler/
  https://repl.it/languages/python3
- Sur un téléphone ou une tablette : installer une application

Ecrire un programme en Python

Deux fenêtres. Un Editeur (pour taper du code) et une Console (donne le résultat)

1) Editeur. le code = un texte contenant les instructions .
Run exécuter le programme. Python lit le texte et suit les instructions.

2) Console. Les résultats ou les messages d'erreurs s'affichent.

En Pyhthon :

- Les commentaires commencent par le symbole #
- Les blocs d'instructions sont identifiés par l'indentation
- Des mots sont réservés à Pyhton :
  if, else, while, def, int, string, bool, print, input....

Exemple :

Python

```
a= 5
def Calcul(n): # Ceci est un commentaire
  if n<2 :
    return 1
  else:
    return (n*9 -2)
b=2+a
c =Calcul(b)
print(c)
```

petiteboitevioletprint() est une fonction Python qui affiche quelque chose dans la console.

`print(♡)` :

- Si ♡ variable, affiche le contenu de la variable
- Si ♡ chaine de caractère, affiche le texte
- Si ♡ opération, affiche le résultat de l'opération

`print(♡, ♠, ♢)` affiche plusieurs éléments.

♠ = input(♡) affiche le message ♡ dans la console et attend.
Il faut entrer quelque chose au clavier et valider.
Ce qu'on a tapé au clavier est alors stocké dans la variable ♠ au format string.

Remarque : La variable créée par input est de type *string*. Si on veut un nombre, il faut la convertir.

- `int(input("Entrer un nombre "))` conversion en entier
- `float(input("Entrer un nombre "))` conversion en nombre décimal

Ecrire le programme dans un éditeur permet de faire des instructions plus longue qu'une ligne, de sauvegarder, modifier plus facilement et organiser son programme.

Plan

Définition de variable

Définition

Une variable est un espace de stockage dans un emplacement de la mémoire (de l'ordinateur).

Caractéristiques :

- nom : permet d'identifier la variable
- type : définit quelle donnée peut contenir la variable
- valeur : utilisée par le programme, peut éventuellement changer

Remarque : Python gère souvent tout seul les types de variables

Définition

Déclarer une variable : attribuer un nom et un type à une boite mémoire

Affecter une variable : associer une valeur à la boite

En Python, on donne le nom [=] la valeur et le type est déduit automatiquement (dans les cas simples) :

NomDeVariable [=] *ValeurDeVariable*

Crée une variable appelée *NomDeVariable* (si elle n'existe pas encore) et met dedans la valeur *ValeurDeVariable*.

Exemple :

Python

```
A= 50
prix = 1.52
Total = A * prix
b= "prix de l'essence"
```

A, b, prix et Total sont des variables. A est un `int`, prix et Total des `float` et b un `string`

Propriété.

Bonnes pratiques pour nommer une variable :

- Commencer par une lettre minuscule
- Choisir un nom explicite pour bien comprendre le programme
- Utiliser des majuscules au milieu du nom _ pour séparer des parties du nom
- Eviter les lettres accentuées mais chiffres autorisés

Remarque : majuscules $\neq$ minuscules.

prix $\neq$ Prix

Remarque : Python permet d'affecter plusieurs variables à la fois.

$$a, b = 1, "bonjour"$$

met directement 1 dans la variable a et "*bonjour*" dans la variable b.

Donc pour échanger les valeurs de a et b, il suffit de faire

$$a, b = b, a$$

Fonctions utiles

- `type(♡)` donne le type de la variable ♡
- `print(♡)` affiche le contenu de la variable ♡.

Plan

Types de nombres

Les nombres peuvent être de différents types :

- int entiers relatifs
- float flottant (nombres décimaux).
  Ils s'écrivent avec un point à la place de la virgule.
- complex Nombres complexes de la forme $a + bj$.
  - La partie imaginaire se note avec un j
  - a et b sont des float.
  - obligatoire de mettre un nombre devant le j, même 0 ou 1.

Exemple :

- -6.9 float
- -69 int
- 3 int
- 3.0 float
- 2+5.1j complex
- -9.7 + 0j complex

Conversion de type

Il est possible de convertir quelque chose en un type de nombre.

- `int(♡)` converti ♡ en entier
- `float(♡)` converti ♡ en nombre flottant.
- `complex(♡, ♠)` crée le nombre complexe $♡ + i♠$

Exemples :

- int(3.6) $\rightarrow$ 3
- float(5) $\rightarrow$ 5.0
- complex(2.3)$\rightarrow$ 2.3 + 0j
- complex(2.3, 6)$\rightarrow$ 2.3 + 6.0j

Remarque : Les trop petits nombres ($< 10^{-323}$) deviennent 0.
Les nombres trop grands ($> 10^{308}$) renvoient une erreur.

Opérations

- $+$ addition : $3 + 4 \rightarrow 7$
- $-$ soustraction : $5 - 6.7 \rightarrow -1.7$
- $*$ multiplication : $12 * (-4)$
- $/$ division avec résultat à virgule : $81/10 \rightarrow 8.1$
- $//$ quotient de la division euclidienne $81//10 \rightarrow 8$
- $\%$ reste de la division euclidienne $81\%10 \rightarrow 1$
- $**$ Puissance : $2 * *3 \rightarrow 2^3$, on peut aussi utiliser pow(2,3)

Remarque : entiers et flottants compatibles pour les opérations arithmétiques, mais résultat flottant.

Pour faire d'autres fonctions, il faut charger une bibliothèque mathématique

`from math import *`

- racine carrée `sqrt(♡)`
- π `pi`
- e `e`
- fonction exponentielle `exp(♡)`
- fonction ln `log(♡)`
- fonctions trigonométriques `cos(♡)`, `sin(♡)`, `tan(♡)`

imprécision avec les float

```
x=0.1
y=x+x
print("y est-il égal à
0.2 ?" , y==0.2)
z=x+x+x
print("z est-il égal à
0.3 ?" , z==0.3)
print("z=", z)
```

y est-il égal à 0.2 ? True
z est-il égal à 0.3 ? False
z= 0.30000000000000004

Les float en Python sont stockés d'une manière "non exacte" en mémoire, et Python affiche une valeur approchée.

Exemple : 0.1 est stockée en python avec une valeur de

0.1000000000000000055511151231257827021181583404541015625

Attention test d'égalité avec des float.

Mode fraction

```
from fractions import Fraction
```

- `Fraction(♡,♠)` $\frac{♡}{♠}$ exactement.
- pour convertir un float ♠ en fraction `Fraction('♠')`

Mode décimal (plus précis mais plus gourmand en mémoire)

```
from decimal import Decimal
```

Pour écrire un nombre en décimal, `Décimal('♠')`

Plan

Chaine de caractères

Définition

Une chaine de caractère, type string (str), s'écrit entre des "guillemets " (ou des 'guillemets').

Exemple : "Bonjour" est un string
"landfql r gfmk " aussi
"3" aussi

Opérations

- $\backslash n$ dans une chaine de caractère insère un retour à la ligne
- $+$ concatène deux chaines de caractère : "chaine1" + "chaine2" donne "chaine1chaine2"
- ♡[n] donne la lettre numéro n de la chaine de caractère ♡, en comptant à partir de 0.
- ♡[n :m] extrait de ♡ la sous-chaine allant du caractère numéro *n*(inclus) au caractère numéro m (exclus), en comptant à partir de 0.
- print(♡) affiche la chaine de caractère
- len(♡) donne la longueur de la chaine de caractère
- str(♡) convertit un objet en chaine de caractère.

Exemple : "Bonjour" + "le monde" donne "Bonjourle monde"
"Bonjourle monde"[4] donne la lettre o.

Plan

Booléen

Définition

Un booleen `bool` ne peut prendre que deux valeurs

`True` (vrai) ou `False` (faux)

Par défaut, la majorité des objets Python sont considérés comme `True`.

- Les chaines de caractères non vides sont True. Une chaine de caractère vide est False.
- Un nombre non nul est True. Le nombre 0 est False.
- Une liste non vide est True. Une liste vide est False.
- La valeur `None` est False

Les booléens servent à écrire des conditions et des tests.

Condition

Définition

Une condition est une opération qui renvoie un booléen `True` ou `False`.

Comparaisons

1. $==$ (est égal) True si les deux valeurs sont égales et False sinon
2. $!=$ (est différent) True si les deux valeurs ne sont pas égales et False sinon
3. $>$ (est strictement supérieur) True si la première valeur est strictement plus grande que la deuxième et False sinon
4. $>=$ (est supérieur ou égal) True si la première valeur est plus grande ou égale à la deuxième et False sinon
5. $<$ (est strictement inférieur) True si la première valeur est strictement plus petite que la deuxième et False sinon
6. $<=$ (est inférieur ou égal) True si la première valeur est plus petite ou égale à la deuxième et False sinon

Exemple : Pour $x = 3$ et $y = 7$:

- $x == 2 \rightarrow$ False
- $y! = 5 \rightarrow$ True
- $x > 1 \rightarrow$ True
- $y <= x \rightarrow$ False

Opérations de booléens

- (*Condition*1) and (*Condition*2) : True si les deux conditions sont vraies, et False sinon
- (*Condition*1) or (*Condition*2) : True si une des deux conditions est vraie, et False sinon
- not(*Condition*1) est True si (*Condition*1) est False. Et inversement.

Exemple : Pour $x = 3$ et $y = 7$,

- $(x == 3)$ and $(y == 7) \rightarrow$ True
- $(x > 1)$ and $(y > 10) \rightarrow$ False
- $(x > 1)$ or $(y > 10) \rightarrow$ True
- not$(x == 1) \rightarrow$ True.

Plan

Définition

Une instruction conditionnelle est une instruction qui dépend d'une condition. Certaines parties du code sont exécutées ou ignorées selon que la condition est vérifiée ou non

if *Une condition* :

```
instructions à faire si la condition est True
avec indentation obligatoire
```

else :

```
instructions à faire si la condition est False
Facultatif, mais indentation obligatoire
```

Exemple : On veut créer un suite d'instruction qui teste si un nombre A est pair, puis qui calcule $A/2$ si c'est le cas et $A+1$ si A est impaire. Pour savoir si A est pair, on teste si le reste de sa division euclidienne par 2 `A % 2` est nul.

Python

```
A=
if (A% 2)== 0 :
   A=A/2
else :
   A= A+1
print(A)
```

Si on rentre $A = 27$ au début, le programme affiche 28.
Si on rentre $A = 26$ au début, le programme affiche 13.0.

Remarque : Si... alors... sinon... permet de séparer deux cas. Si on veut faire plus de cas, on peut imbriquer plusieurs **Si** (ou utiliser d'autres structures).

Exemple : Comparer deux nombres x et y. Stocker dans une variable max le plus grand des deux si il sont différents. Afficher un message si il sont égaux.
Il y a trois cas : $x < y$, $x = y$ et $x > y$.

Python

```
if (x==y):
  print("Les valeurs sont égales")
else :
  if (x>y):
    max= x
  else :
    max= y
```

Si on teste avec $x = 6$ et $y = 2 * 3$, on voit "Les valeurs sont égales".
Si on teste avec $x = 95$ et $y = 76$, alors *max* prend la valeur 95

Plutôt que d'utiliser des si-alors-sinon imbriqués, on peut utiliser `elif` (raccourci de else if)

Python

```
if (x==y):
  print("Les valeurs sont égales")
elif (x>y) :
  max= x
else :
  max= y
```

Il reste quoi ?

1. Quelle fonction permet d'afficher quelque chose dans la console ?
2. la variable $a = 3.0$ est de quel type ?
3. Que renvoie "*Chaise*"[2] ?
4. if x==2 : x= x+3 else x= x-1 donne quoi si $x = 4$?

Il reste quoi ?

1. Quelle fonction permet d'afficher quelque chose dans la console ? print()
2. la variable $a = 3.0$ est de quel type ? Float
3. Que renvoie "*Chaise*"[2] ? la lettre *a*
4. if x==2 : x= x+3 else x= x-1 donne quoi si $x = 4$? x=3