Créer un programme

1 - Structure et exécution d'un programme

1.1 Différence entre console interactive et programme

Cas de la console

Lorsqu'une ligne contient **uniquement une expression**, l'interpréteur l'**évalue et affiche** le résultat.

```
>>> a = 10
>>> b = a + 2
>>> b
12
```

Cas du programme

Lorsqu'une ligne contient **uniquement une expression**, l'interpréteur l'évalue et... c'est tout. **Pas d'affichage**.

```
a = 10
b = a + 2
b
```

Dans un programme, si vous voulez voir quelque chose s'afficher, il faut le demander explicitement :

```
a = 10
b = a + 2
print(b)
```

1.2 COMMENTAIRES : expliquer le fonctionnement

Les commentaires sont destinés à un lecteur humain et ils visent à rendre **le code interne facile à comprendre**. L'interpréteur Python ne tentera pas d'exécuter les commentaires.

— **Commentaire sur toute une ligne**

```
# Toute cette ligne est un commentaire.
```

— **Commentaire en fin de ligne**

```
print("Bonjour tout le monde") # Ceci est également un commentaire
```

— **Cette ligne ne comporte aucun commentaire puisque le # fait juste partie d'un string.**

```
print("Cette ligne ne contient pas de # commentaire")
```

1.3 Structure d'un programme

— **Importation des modules nécessaires**.

— **Déclarations des CONSTANTES**.

— **Déclaration des variables globales** destinées à être lue depuis des fonctions : à utiliser avec modération.

— **Déclaration des fonctions**.

— **Les instructions du programme en lui-même** : on nomme cette partie "programme principal" parfois. On y place également les variables globales qu'on envoie simplement en tant que paramètres.

Convention On sépare les parties par au moins deux lignes vides.

1.4 Noms des variables globales et locales

Deux utilisations des variables globales (voir le site) :

— **lues directement par les fonctions**. Pratique mais si vous changez le nom dans le programme, c'est l'ensemble des lignes de vos fonctions qu'il faut modifier.

— **transmises aux fonctions en tant que paramètres**. Avantage : le nom de la variable globale n'a pas d'importance.

1.5 Evitez les variables globales lues directement

Trois raisons d'éviter la lecture directe ?

— Votre fonction n'est pas réutilisable dans un autre programme, à moins de d'utiliser les mêmes noms de variables.

— Le moindre changement de nom de votre variable globale provoquerait l'échec de toutes fonctions qui utiliseraient l'ancien nom.

— Dans un grand programme, quelqu'un pourrait avoir l'idée de créer une nouvelle variable globale qui aurait malheureusement le même nom que la votre.

2 - IHM console

2.1 IHM Console

Le disposif qui permet de faire interagir deux systèmes sans connaître les détails de fonctionnement de l'autre se nomme une **interface**. Lorsque l'interface se fait entre un homme et une machine, on la nomme **IHM : Interface Homme Machine**.

L'IHM-console gère l'interaction avec `print()` et `input()`.

2.2 `print()`

Affichage avec `print()`

```
>>> a = "Salut à toi !"
>>> print(a)
  Salut à toi !
```

Aucune mémoire avec `print()`

On ne **mémorise pas** l'information affichée avec `print()`.

Exemple explicite : on récupére dans `rep` la réponse de la fonction `print()`. rep ne contient rien.

```
>>> a = "Reçu ?"
>>> rep = print(a)
  Reçu ?

>>> rep
>>>
```

Que peut-on envoyer comme entrée à `print()` ?

Tout ce que vous voulez.

Plusieurs arguments

On peut envoyer plusieurs arguments en les séparant par des virgules.

```
1  c = 15
2  print("Var c : ", c)
```

```
Var c : 15
```

`f-string`

On utilisera plutôt un **f-string**. C'est un string :

- Précédé d'un `f`
- Pouvant contenir des **expressions entre accolades** qui seront évaluées puis affichées.

```
1  c = 15
2  print(f"Var c : {c}")
```

```
Var c : 15
```

2.3 print() et caractères de contrôle

Les caractères de contrôle n'affichent rien mais provoquent un effet) : le passage à la ligne `/n`, la tabulation `/t`...

Pour provoquer l'effet voulu, il faut utiliser `print()`.

Pour connaitre le code UNICODE d'un caractère, il faut utiliser `ord()`. Signature : `ord(str) -> int`

Pour connaître le caractère correspondant à un numéro, il faut utiliser `chr()`. Signature : `chr(int) -> str`

2.4 Un print() n'est pas un return !

La fonction `print()` ne renvoie rien. Il faut lire l'énoncé :

- fonction qui renvoie : mot-clé `return`
- fonction qui affiche : fonction `print()`

2.5 `input()`

Principe fondamental

La fonction native `input()` récupère la réponse tapée au clavier **sous forme d'un string, même les nombres**. Pour valider la réponse, il faut appuyer sur entrée .

```
>>> note = input()
18

>>> note    >>> note * 2
'18'        '1818'
```

Récupérer un entier ou un flottant

Il faut tenter de convertir le string reçu en integer, en utilisant la fonction native `int()` ou `float()`.

```
>>> reponse = int(input())
5

>>> reponse  >>> reponse * 3
5            15
```

Utilisation complète

On peut aussi lui transmettre un string contenant la question.

```
reponse = input("Veuillez
fournir un nombre entier : ")
```

Python va alloir faire trois choses :

- Afficher votre message mais **sans passage à la ligne**.
- Récupérer la réponse **sous forme d'un string**
- Placer ce string dans `reponse`

```
>>> reponse = int(input("Note ? "))
Note ? 5

>>> reponse
5
```

Et en cas d'impossibilité de conversion ?

HP. Voir le site pour `try...except`.