Université Paris-Saclay
M2 Droit de la création et du numérique

Approche de l'élaboration et du fonctionnement des logiciels

Alain Delaët

objectif des séances 2 & 3

faire l'apprentissage de la programmation avec le langage Python

on propose un environnement de travail dans le navigateur

`https://jupyterhub.ijclab.in2p3.fr/`

1. cliquer sur le bouton `JupyterLab`
2. créer un dossier (par exemple `td2`) et s'y déplacer

deux façons de travailler

- un **mode interactif** $\approx$ comme une calculatrice
  - on saisie une instruction
  - l'interprète Python l'exécute et affiche le résultat
- un **mode programme**
  - on édite un fichier Python contenant une séquence d'instructions
  - on le fait exécuter par l'interprète Python
  - seules nos instructions `print` donnent lieu à un affichage

mode interactif

- cliquer sur le bouton + (Nouveau lanceur) et choisir `Console->Python 3`
- saisir une commande Python et utiliser `Shift+Entrée` pour l'exécuter
- recommencer

```
>>> 1+2
3
>>>
```

```
>>> 2 + 5 * (10 - 1 / 2)
49.5
>>>
```

les espaces sont purement décoratifs et ne modifient pas la façon dont l'expression est calculée

```
>>> 1+2 * 3
7
```

l'interprète Python rejette toute instruction mal formée

```
>>> 1 + * 2
File ..., line 1
1 + * 2
^
SyntaxError: invalid syntax
```

il n'y a pas eu d'exécution

erreur pendant l'exécution

l'exécution peut également échouer, ici sur une division par zéro

```
>>> 2 / (3 - 3)
Traceback (most recent call last):
File ..., line 1, ...
ZeroDivisionError: division by zero
```

le résultat d'un calcul peut être stockée dans une **variable**

```
>>> a = 1 + 1
>>>
```

la notation `a =` est une **affectation** : le résultat de 1+1 est mémorisé dans la variable `a`

```
>>> a
2
```

on peut maintenant utiliser la variable `a` dans d'autres calculs

```
>>> a * (1 + a)
6
```

peut être formé de plusieurs caractères, chiffres et soulignés

```
>>> cube = a * a * a
>>> ma_variable = 42
>>> ma_variable2 = 2019
```

mais ne peut pas commencer par un chiffre

```
>>> 4x = 2
File ..., line 1
4x = 2
^
SyntaxError: invalid syntax
```

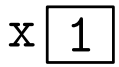
variable non définie

il n'est pas possible d'utiliser une variable qui n'a pas encore été définie

```
>>> b + 1
Traceback (most recent call last):
File ...., line 1, in <module>
NameError: name 'b' is not defined
```

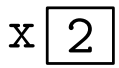
il faut penser une variable comme une petite **boîte**

```
>>> x = 1
```



et une affectation comme une modification de son contenu

```
>>> x = x + 1
```



(la valeur **varie** avec le temps, d'où le nom de variable)

pour écrire un programme formé de plusieurs instructions, on les écrit **dans un fichier** (avec le suffixe `.py`) que l'on donne à l'interprète Python
les valeurs calculées ne sont plus affichées
il faut utiliser l'instruction `print` de Python si on veut que le programme affiche quelque chose

- cliquer sur le bouton + (Nouveau lanceur) et choisir `Autre->Fichier Python`
- le renommer (bouton droit ou F2), par exemple `test.py`
- éditer et sauvegarder
- pour exécuter le programme,
  - cliquer sur le bouton + (Nouveau lanceur) et choisir `Autre->Terminal`
  - dans le terminal, taper la commande `python test.py`
- modifier le programme si besoin, sauvegarder et exécuter de nouveau (dans le terminal, utiliser la flèche vers le haut pour récupérer la dernière commande)

dans le fichier `test.py` on écrit une **séquence** de quatre instructions

```
x = 6 * 7
print(x)
x = x + 1
print(x)
```

puis on l'exécute, ce qui a pour effet d'évaluer chaque instruction, dans l'ordre du fichier

```
$ python test.py
42
43
```

la commande `print` n'est pas limitée à des nombres
on peut lui donner un message, entre guillemets

```
print("Bonjour !")
```

le texte entre guillemets s'appelle une **chaîne de caractères**

Python n'interprète pas le contenu d'une chaîne de caractères ainsi,

```
print("1 + 3")
```

affiche

```
1 + 3
```

on peut récupérer des caractères tapés au clavier avec `input`

```
s = input()
```

la variable `s` est une chaîne de caractères
on peut la convertir en entier avec l'instruction `int`

```
a = int(s)
print("le nombre suivant est ", a + 1)
```

```
# calcul de l'âge
saisie = input("Entrez votre année de naissance : ")
annee = int(saisie)
age = 2048 - annee   # on calcule l'âge par soustraction
print("Vous aurez", age, "ans en 2048.")
```

tout ce qui est au-delà de # est ignoré par Python ; c'est un **commentaire**

boucles

pour l'instant, le programme exécute autant d'instructions qu'il y a de lignes dans le fichier Python
pour réaliser un programme plus complexe, on peut **répéter** plusieurs fois les mêmes instructions ; on appelle cela une **boucle**

on peut exécuter trois fois la même instruction avec une boucle `for`

```
for _ in range(3):
    print("hello")
```

l'instruction à répéter est **en retrait** ; c'est le corps de la boucle

on peut répéter plusieurs instructions

```
p = 1                    # une fois
for _ in range(10):
    print(p)             # dix fois
    p = 2*p              # dix fois
print("c'est fini")      # une fois
```

une fois les 10 tours de boucle effectués, on poursuit avec les instructions suivantes (ici `print("c'est fini")`)

reprenons le programme du cours 1
on peut l'écrire plus simplement avec une boucle `for`

```
change = 0 # score du joueur qui change de porte
reste  = 0 # score du joueur qui ne change pas
for _ in range(1000):
  g = randint(1, 3) # la porte gagnante
  c = randint(1, 3) # la porte choisie par le joueur
  if c == g:
    reste = reste + 1
  else:
    change = change + 1
print("le joueur qui change gagne", change, "fois")
print("le joueur qui reste  gagne", reste , "fois")
```

```
n = int(input("combien de nombres ? "))
somme = 0

for _ in range(n):
  x = int(input())
  somme = somme + x

print("la somme vaut", somme)
```

le programme

```
for i in range(10):
  print(i)
```

va afficher les dix lignes

```
0
1
2
...
9
```

la variable `i` est appelée l'**indice de boucle**
attention : les valeurs de `i` vont de **0** à **9**

calcul des intérêts à 2% sur 10 ans

```
b = 1000
for i in range(10):
  b = b + 0.02 * b
  print("après", i+1, "années :", b)
```

```
après 1 années : 1020.0
après 2 années : 1040.4
après 3 années : 1061.208
...
```

un peu de hasard

comme tous les langages, Python offre un **générateur de nombres pseudo-aléatoires**

ce n'est pas aléatoire au sens mathématique, mais suffisant en pratique pour beaucoup de choses (des jeux, par exemple)

≪ suffisant ≫ veut dire ici qu'il est très difficile de prédire le prochain nombre de la séquence

on utilise une **bibliothèque Python** (un ensemble de programmes déjà écrits), qui s'appelle `random`, et plus spécifiquement la **fonction** `randint` qu'elle fournit

```
from random import randint
```

exemple :
`randint(1, 6)` nous donne un nombre aléatoire entre 1 et 6 inclus

on simule 20 lancés d'un dé à 6 faces

```
from random import randint

for _ in range(20):
    print(randint(1, 6))
```

tests

dans un programme, il est fondamental de pouvoir **tester une condition** puis de poursuivre deux exécutions différentes selon le résultat

- si la condition est vraie, faire ceci
- sinon, faire cela

```
if score == 51:
  print("Victoire")
else:
  print("la partie continue...")
```

on appelle cela une instruction **conditionnelle**

le test d'égalité s'écrit ==

le symbole = est réservé à l'affectation

`x != y`	différent
`x > y`	plus grand
`x >= y`	plus grand ou égal
`x < y`	plus petit
`x <= y`	plus petit ou égal

on peut utiliser `and` (pour « et ») et `or` (pour « ou »)
exemple :

```
if n < 1 or n > 100:
  print("ce n'est pas entre 1 et 100")
```

```
if score < 10:
    print("pas mal...")
elif score < 50:
    print("très bien")
else:
    print("excellent !")
```

```
from random import randint

n = randint(1, 6)

c = int(input("Votre choix (entre 1 et 6) ? "))
if c < 1 or c > 6:
    print("Merci de respecter la consigne")
elif c == n:
    print("Vous avez gagné !")
else:
    print("Vous avez perdu...")
```

boucle tant que

on peut exécuter des instructions **tant qu'**une condition est vérifiée

```
while b < 1000:
    b = 1.02 * b
```

on l'avait déjà croisée

```
n = 1000
while n < 1000:
    ...
    n = n+1
```

la suite de Fibonacci, en s'arrêtant à 1000

```
a, b = 0, 1
while a < 1000:
    print(a)
    a, b = b, a+b
print("et voilà !")
```

(remarque en passant : on peut affecter plusieurs variables simultanément)

si on écrit

```
while True:
    ...
```

c'est-à-dire « tant que la condition vraie est vraie »,
alors la boucle ne s'arrêtera jamais

utile pour un programme comme un jeu, dont la durée n'est pas bornée

```
while True:
    ...
    if score == 51:
        print("Victoire !")
        exit() # termine le programme
        ...
```

ici, la boucle infinie sera interrompue par l'instruction `exit()`

Python nous offre

- des entiers, avec des opérations arithmétiques
- des variables pour mémoriser le résultat de calculs

```
age = 2048 - annee
```

- une instruction `print` et des chaînes de caractères

```
print("Vous aurez", age, "ans en 2048.")
```

- des instructions conditionnelles avec `if` / `elif` / `else`
- des boucles bornées (`for`) et tant que (`while`)
- du « hasard » avec `randint`

écrire un programme pour jouer à « devine mon nombre »

1. l'ordinateur choisit au hasard un nombre entre 1 et 100
2. tant que la partie n'est pas finie
   - 2.1 demander un nombre à l'utilisateur
   - 2.2 si c'est le bon, afficher `Bravo !` et terminer le programme
   - 2.3 sinon, afficher `"trop grand"` ou `"trop petit"`

note : on peut choisir

- une boucle `for`, en limitant le nombre d'essais
- une boucle `while`, tant que le nombre n'a pas été trouvé

on pourra améliorer en

- affichant le nombre total d'essais
- affichant un message si le nombre entré n'est pas entre 1 et 100
  (et ne pas compter l'essai dans ce cas)
- affichant un message plus précis
  (`"beaucoup trop grand"` / `"un peu trop grand"` / etc.)

variantes :

- faire plutôt jouer le programme
- **plus difficile** : faire ≪ tricher ≫ le programme,
  c'est-à-dire maximiser le nombre de tentatives du joueur

- **vendredi 14 février 14h en salle B-109**
  - programmation Python 2/2