Open Minds Club

Formation

Langage Python

Sidahmed Bilal Malaoui

14es Février 2017 — 20 Mars 2017

Sommaire

1 La programmation avec Python 3 **1**

1.1 Introduction . . . 1
1.1.1 C'est quoi un programme ?? . . . 1
1.1.2 Comment sont crée ces programmes ?? . . . 2
1.1.3 Les langages de programmation les plus connus . . . 3
1.1.4 Compilé vs Interprété . . . 4
1.1.5 Le langage Python . . . 4
1.2 Les logiciels nécessaires pour la programmation . . . 5
1.2.1 Installer Python et PyCharm sous Ubuntu Linux . . . 6
1.2.2 Installer Python et PyCharm sous Windows . . . 7
1.3 Hello World . . . 9
1.3.1 Les commentaires . . . 11
1.3.2 Exercice 1 . . . 11
1.3.3 Solution de l'exercice . . . 12
1.4 Les variables . . . 13
1.4.1 Créer et utiliser les variables avec python . . . 14
1.4.2 Quel genre d'information on peut stocker dans nos variables 15
1.4.3 Afficher les variables . . . 15
1.4.4 Récupérer une saisie . . . 16
1.4.5 Comment nommé les variables . . . 17
1.4.6 Exercice 2 . . . 18
1.5 les opérateurs arithmétique . . . 19
1.5.1 Addition, soustraction, multiplication . . . 19
1.5.2 Division entière et Division euclidienne . . . 19
1.5.3 Modulo (reste de la division euclidienne) . . . 19
1.5.4 La puissance . . . 20
1.5.5 Opérations entre variables . . . 20
1.5.6 Bonus . . . 20
1.5.7 Un petit problème ! . . . 21
1.5.8 Exercice 3 . . . 21
1.6 Les conditions . . . 23
1.6.1 Le if . . . 23
1.6.2 Le else . . . 24
1.6.3 Le elif . . . 24
1.6.4 Comparez les chaînes de caractères . . . 25
1.6.5 Plusieurs conditions à la fois . . . 25
1.6.6 Les booléens . . . 26
1.6.7 Exercice 4 . . . 27
1.7 Les boucles . . . 28
1.7.1 La boucle while . . . 28
1.7.2 La boucle for . . . 29
1.7.3 Un bonus de l'utilisation de la boucle for . . . 30
1.7.4 Exercice 5 . . . 31
1.8 Découper le programme en fonctions . . . 32
1.8.1 C'est quoi une fonction ? . . . 32
1.8.2 Fonction simple . . . 32

1.8.3 Plusieurs paramètres 33
1.8.4 Fonction qui ne renvoie rien 34
1.8.5 Fonction qui renvoie plusieurs valeurs 34
1.8.6 Pourquoi découper notre programme ?? 35
1.8.7 Exercice 6 35

1 La programmation avec Python 3

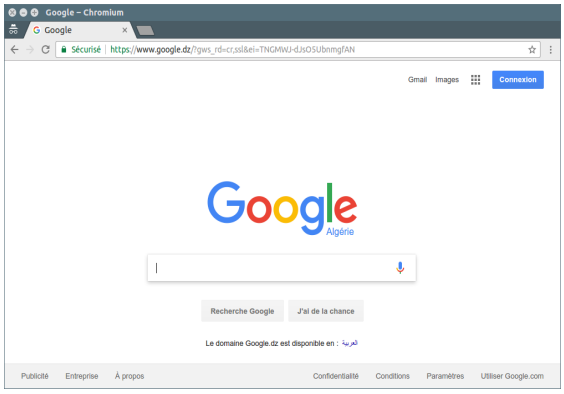
1.1 Introduction

Ceci est une petite introduction où on va parler des programmes et des langages de programmations.

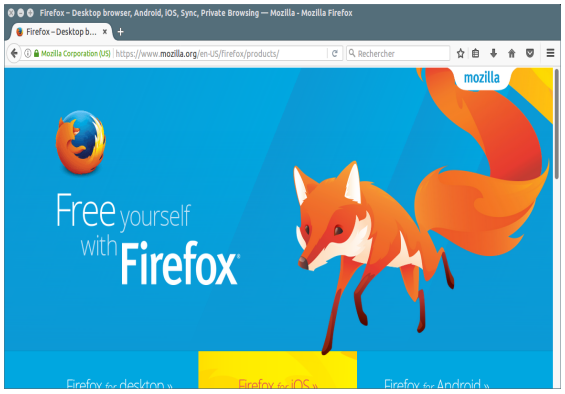
1.1.1 C'est quoi un programme ??

Je vais commencer par donner une définition qui peut vous semblez un peu floue, mais cette description va s'éclaircir coûte que coûte au cours de notre formation.

Alors, un programme est un ensemble d'instructions exécuté par l'ordinateur pour qu'il puisse (l'ordinateur) accomplir une tâche bien précise. On a pour exemple des programmes : les navigateurs web (comme chromium, chrome, firefox), qui permettent de visiter les cites et de chatter en ligne ; on a aussi les jeux vidéos, les logiciels de retouches de photos … etc



(a) Google Chrome



(b) Mozilla Firefox

FIGURE 2 – Le jeu Assassin's Creed

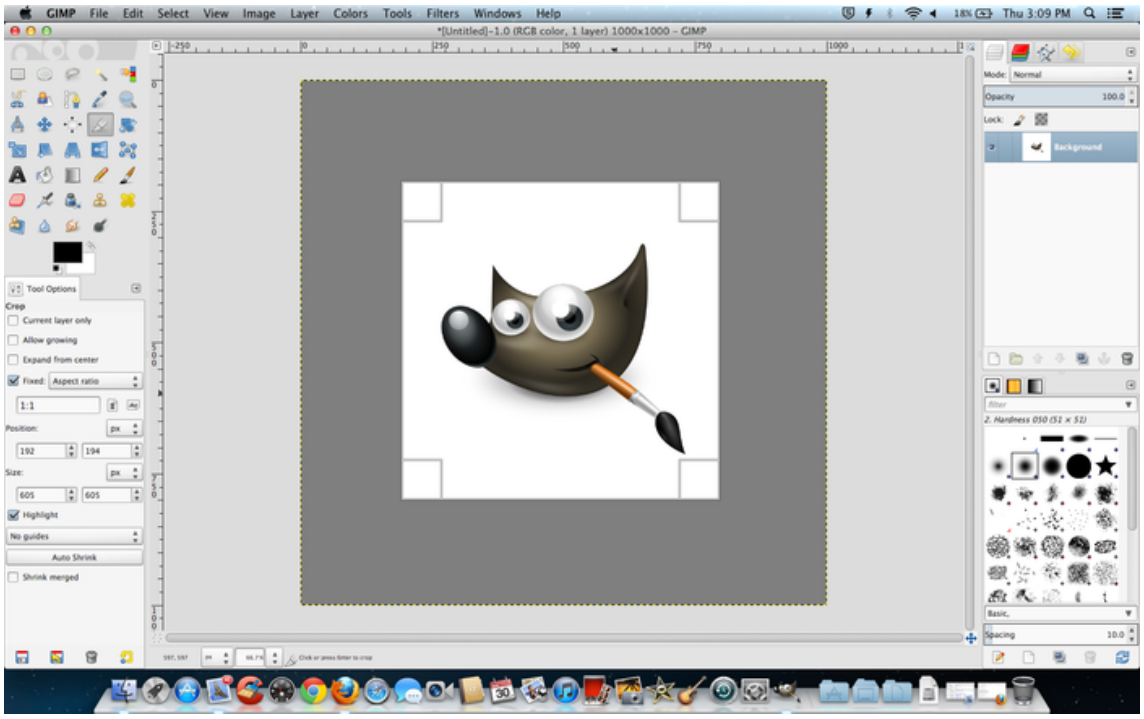


FIGURE 3 – Logiciels de retouches “Gimp”

1.1.2 Comment sont crée ces programmes??

L’ordinateur est une machine bête et discipliné qui ne comprend qu’un seul langage, qui est le langage *binaire*. Ce langage est une succession de 1 et de 0 comme suit :

```
01101100001100110011000111101011101
```

Donc, normalement, pour créer des programmes, il fallait apprendre ce langage. Mais heureusement, les informaticiens ont eu la réflexion de créer d’autres langages intermédiaires (qu’on appel langage de programmation) qui sont beaucoup plus facile que le binaire. Ces langage ont tous le même but : permettre de créer des programmes plus facilement qu’en binaire.

Voici comment cela fonctionne :

- On écrit nos instructions en utilisant un langage de programmation.
- Les instructions sont traduites en binaire grâce à un programme de traduction.
- L’ordinateur peut alors lire le binaire et faire ce qu’on l’a demandé.

Voici une image qui illustre ce qu’on vient de dire :

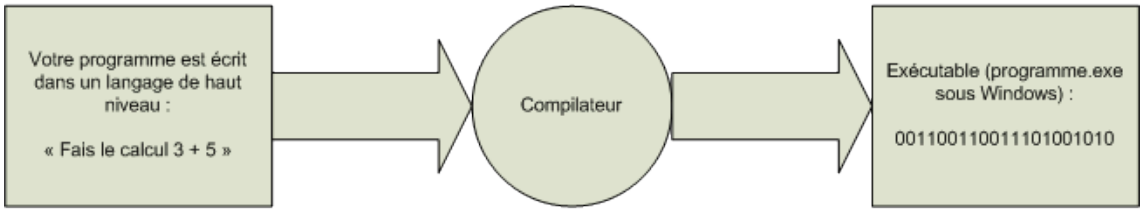


FIGURE 4 – Les étapes de création d'un programme

Une remarque très importante : les instructions que nous écrivons dans un langage de programmation sont appelé **Code Source**. Gardez bien ce terme à l'esprit, parce qu'on va beaucoup l'utiliser par la suite. Voici un exemple de code source python que nous écrirons plus tard :

```python
#!/usr/bin/python3

import os
try:
    import psutil
except ImportError:
    import pip
    print("Installing missing packages (this will be done the first time only) :")
    if os.getuid() == 0:
        pip.main(["install", "psutil"])
    else:
        pip.main(["install", "psutil", "--user"])
    print("Done.\nRerun the command now.")
    exit(0)
import argparse
from time import sleep, localtime


# This function will execute until the end of the copy
def copy(time, verbose):
    while True:
        bytes_read = psutil.disk_io_counters().read_bytes / (1024*1024)
```

FIGURE 5 – Coude source Python

1.1.3 Les langages de programmation les plus connus

C++ : La formation est faite par YASSER.

Java : La formation est faite par WISSAM et SOFIANE.

Python : C'est la formation à laquelle vous assistez maintenant.

(a) Logo C++

(b) Logo Java

(c) Logo Python

1.1.4 Compilé vs Interprété

Nous avons dit tout à l'heure que les instructions sont traduites en binaire pour que l'ordinateur puisse les comprendre et les exécuter. Mais ce qu'on a pas dit c'est qu'il y a deux façons de traduire les instructions : *Compilation* et *Interprétation*.

Compilation : Avec ce processus, toutes nos instructions sont traduites en binaires à la fois. Ce qui nous donne en résultat un gros fichier binaire qui peut être exécuté directement par l'ordinateur.

Interprétation : Avec cette technique, on ne traduit pas tous les instructions en binaires, mais on va plutôt traduire une instruction par instruction à chaque fois qu'on veut exécuter notre programme. Par exemple, si mon code source contient 3 instructions, quand je veux l'exécuter je traduit la première instruction en binaire, et je la passe à l'ordinateur pour qu'il l'exécute ; ensuite je traduit la deuxième instructions, et je la passe à l'ordinateur pour qu'il l'exécute … et ainsi de suite jusqu'à la traduction de toute mes instructions.

Le programme qui fait l'interprétation est appelé l'*interpréteur*, et dans notre cas on va utiliser l'interpréteur *Python*.

1.1.5 Le langage Python

Python est un langage de programmation interprété, *Open Source*, facile à apprendre, et très pratique. On cite quelques avantages de ce langage :

- Python est facile à apprendre et son code est facile à lire.
- Il est multiplate-forme, il marche sous Linux, Windows, et Mac OS.
- C'est un langage orienté objet (on en parlera de ça dans le prochaines séances).
- Il a une bibliothèque tierce qui permet de faire de la programmation web d'une manière très flexible. Cette bibliothèque c'est *Django*.

- Il permet de développer très rapidement en utilisant peu de code, et de crée des applications très complexe avec facilité. Tous ça grâce à son style et à sa bibliothèque standard très complète.
- Il est très utilisé par les pentesters et les hackers.

Pour finir, il faut savoir que les géants de l'informatique comme Google, Yahoo, NASA préférent le langage python ; et ce langage est intégré à la quasi totalité des distributions Linux (C'est-à-dire que vous pouvez l'utiliser dans ces distributions sans devoir l'installer).

Juste pour information, il en existe deux versions de python qui ne sont pas compatibles entres eux. Ces deux versions sont les 2.x et les 3.x . Nous on va utiliser la version 3.x qui est le future de python.

1.2 Les logiciels nécessaires pour la programmation

Pour pouvoir créer des programmes en python, il existe deux manières :

1. Soit on écrit nos instructions dans un fichier texte, et puis on appel l'interpréteur python pour qu'il exécute nos instructions et nous affiche le résultat. Cette méthode comporte de nombreux avantages, et elle très flexible, mais elle est compliqué (surtout pour les débutants). Donc nous allons optez pour la deuxième méthode dans cette formation.
2. Cette méthode consiste à utiliser un programme appelé *IDE* (Integrated Development Environment), ou *EDI* en français (Environnement de Développement Intégré). Ce programme contient tous ce dont on a besoin pour créer nos programmes : une partie ou on peut écrire nos instructions, des boutons pour demander à python d'exécuter nos instructions, et finalement une partie ou le résultat de l'exécution est affiché.

L'IDE qu'on va utiliser est appelé *PyCharm* (Figure 7). C'est un IDE *Open Source* et très puissant.

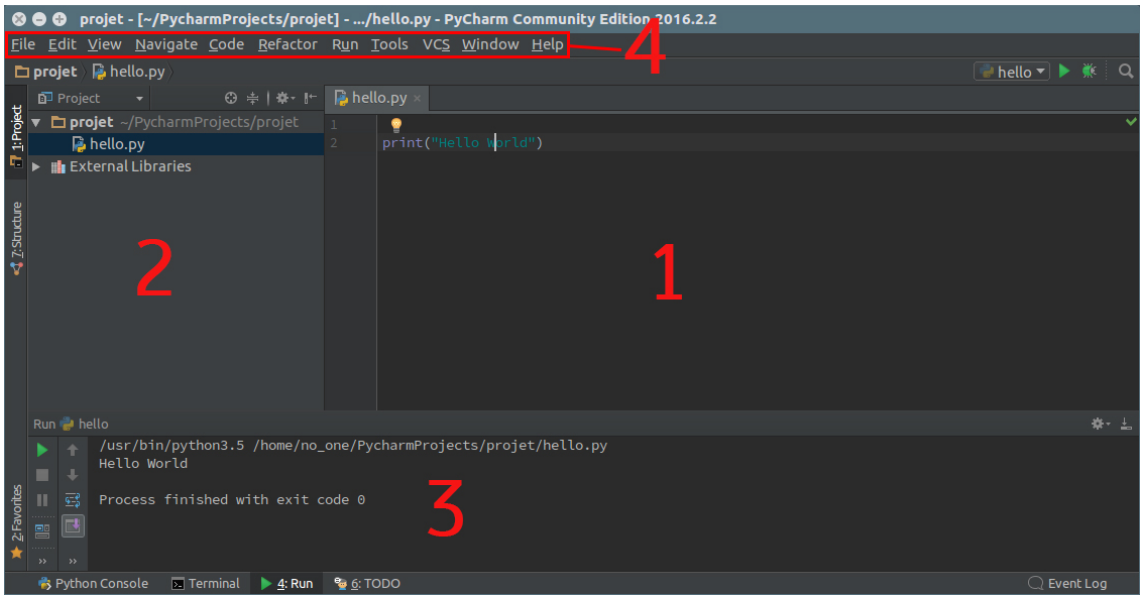


FIGURE 7 – Pycharm

PyCharm est composé essentiellement de 4 partie :

La zone principale (1) : c'est dans cette zone où on écrira notre code source (nos instructions).

La liste des fichiers du projet (2) : dans cette zone on peu voir tous les fichiers de notre projet. On voit bien que le projet présenté dans la Figure 7 contient un seul fichier seulement.

La zone d'affichage (3) : c'est cette zone qui nous affichera le résultat d'exécution de notre programme, où les éventuelles erreurs s'il y en a.

La barre d'outils (4) : elle contient de nombreux boutons qui font différentes choses : exécuter le projet, configurer l'IDE … etc, mais nous on utilisera que quelques boutons.

1.2.1 Installer Python et PyCharm sous Ubuntu Linux

Python est installé par défaut dans toute les distributions Linux. Par contre, l'IDE PyCharm il faut l'installer. Pour cela, il suffit d'ouvrir un terminal (Ctrl+alt+t) et exécuter ces commandes :

```
sudo add-apt-repository ppa:mystic-mirage/pycharm
sudo apt-get update
sudo apt-get install pycharm-community
```

Et voila, PyCharm est installé sur votre machine. Il suffit juste de le lancer en appuyant le bouton windows (Figure 8) et écrire `pycharm` (Figure 9) puis tapez entrer.

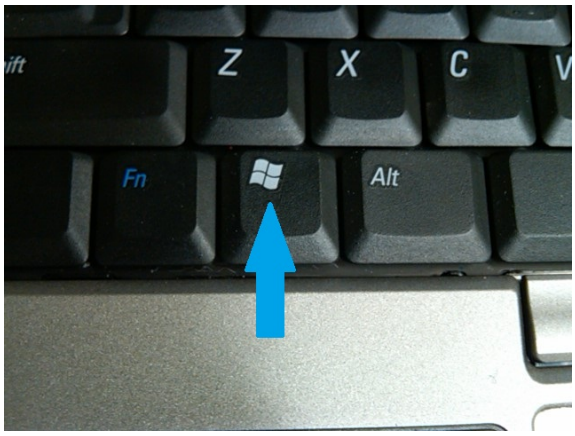


FIGURE 8 – Bouton windows

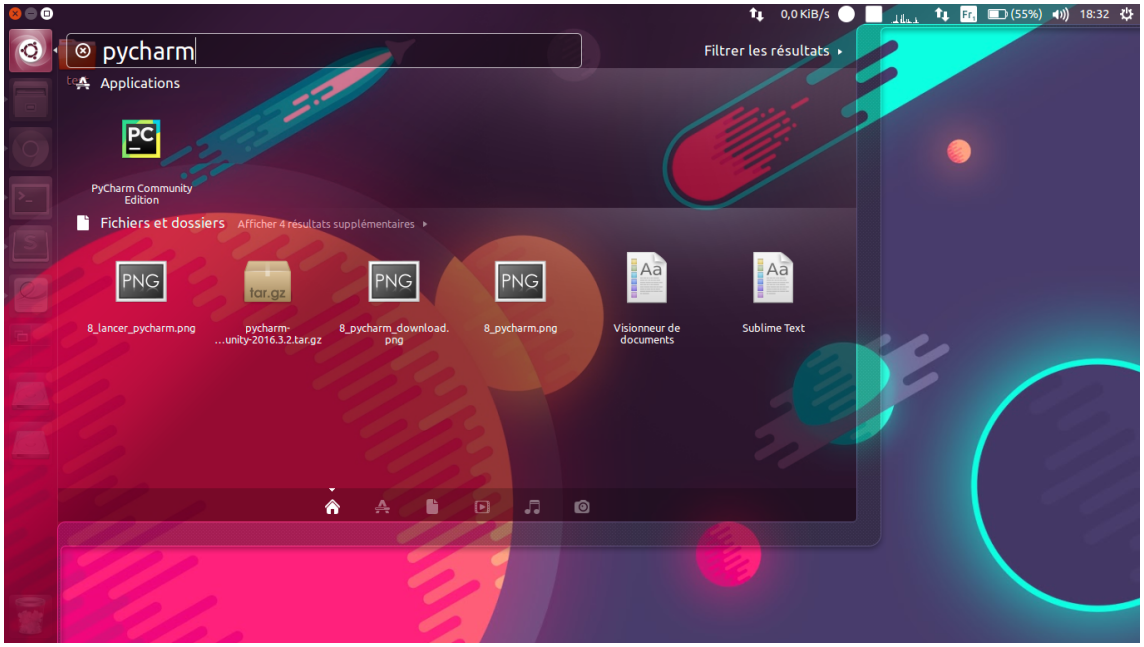


FIGURE 9 – Lancer PyCharm

1.2.2 Installer Python et PyCharm sous Windows

Python n'est pas installé par défaut sur Windows, donc il faut installer Python et l'IDE PyCharm.

Pour installer Python, visitez ce lien https ://www.python.org/downloads/ et télécharger la dernière version de Python (au moment du déroulement de la formation, c'est la version `3.6`). Installez là quand vous finissez le téléchargement.

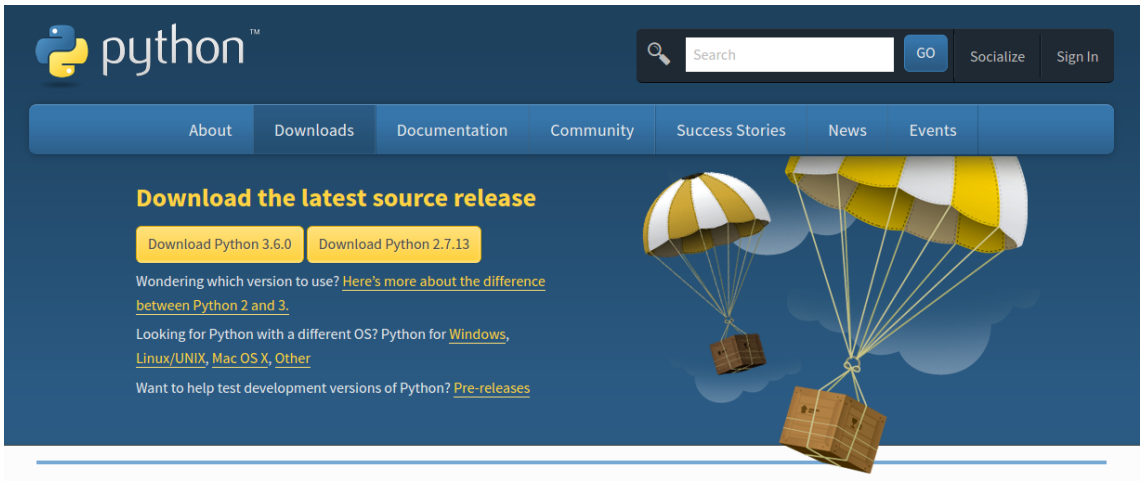


FIGURE 10 – Lien de téléchargement de Python

Pour PyCharm, télécharger la version `community` sur ce lien https ://www.jetbrains.com/pycharm/download/ (Figure 11) et installez là.

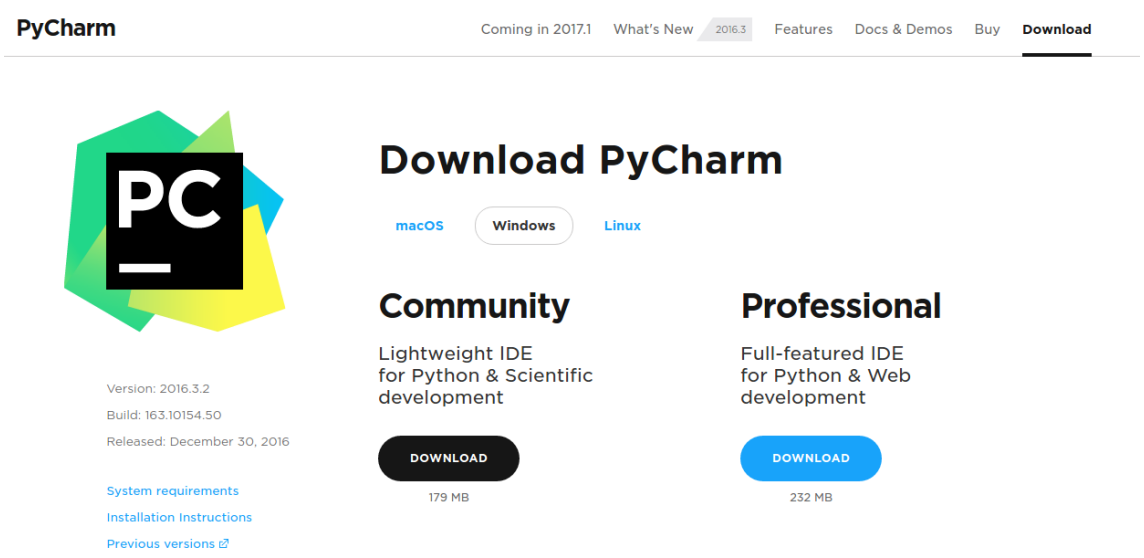


FIGURE 11 – Lien de de téléchargement PyCharm

1.3 Hello World

Nous allons écrire maintenant notre premier programme. Un programme petit et simple pour comprendre le fonctionnement de ce langage fascinant.

On commence par créer un nouveau projet en cliquant sur la barre d'outils sur `File -> New Project`. Ensuite on choisis où on veut placer notre projet, et la version de l'interpréteur qu'on veut utiliser (Figure 12). Choisissez n'importe quelle version 3.x disponible sur votre ordinateur (Pour moi, j'utilise la version 3.5).

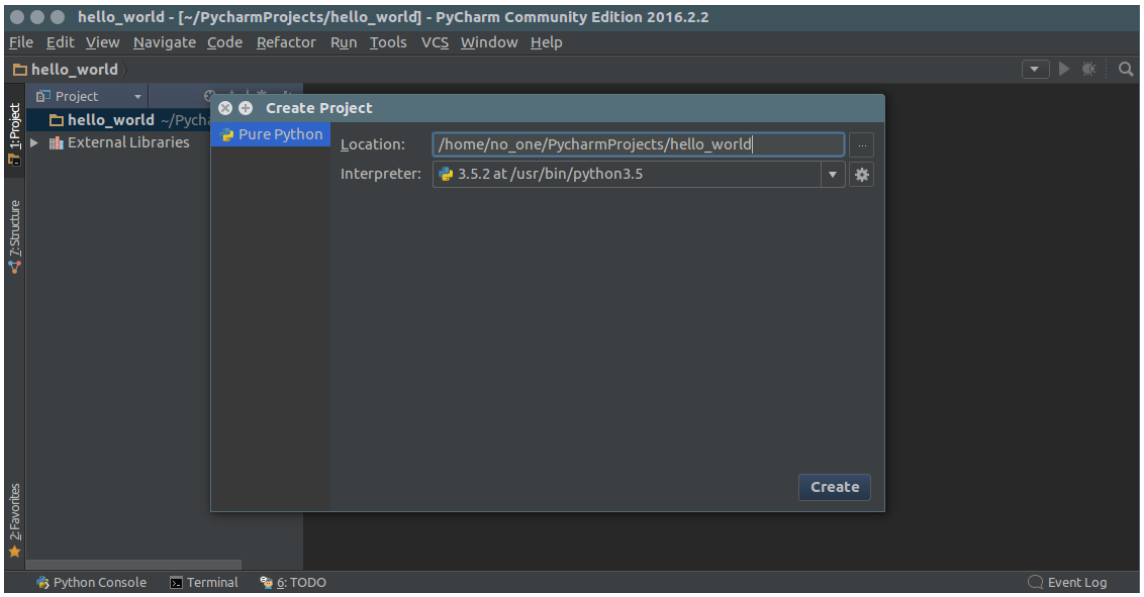


FIGURE 12 – Création d'un nouveau projet

On a crée notre projet, mais ce dernier est vide, il ne contient aucun fichier. Nous allons maintenant ajouter un fichier où nous allons écrire nos instructions (programmes). On fait un clique sur la barre d'outils sur `File -> New` et on choisit `Python File`. On lui donne un nom à notre fichier, et on clique sur `OK` (Figure 13).

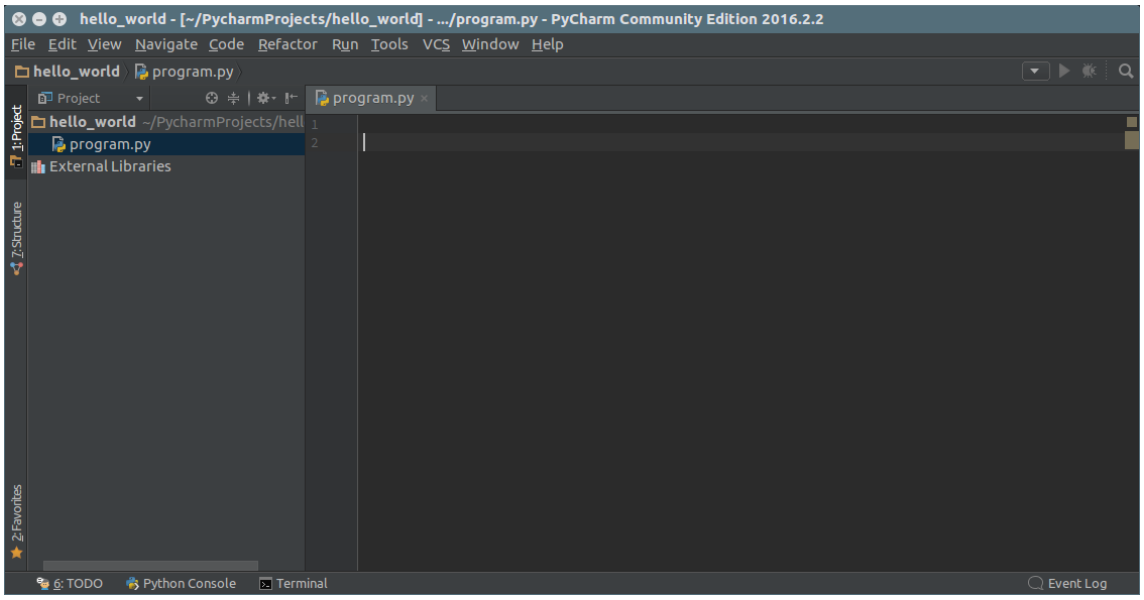


FIGURE 13 – Création d'un fichier pour écrire les instructions

Dorénavant, je vais seulement vous affiché les codes que vous devez écrire dans la zone principale (la zone **1**, Figure 7).

On arrive maintenant à la programmation, voici votre premier programme :

```
1 print("Hello World!")
```

Écrivez ce code source dans la zone principale (zone n° 1 Figure 7), et demandez à PyCharm d'exécuter notre programme en cliquant sur la barre d'outils sur `Run -> Run`, et on choisit le fichier à exécuter (on en a un seul dans notre cas). Voici le résultat de l'exécution :

```
/usr/bin/python3.5 /home/no_one/PycharmProjects/hello_world/program.py
Hello World!

Process finished with exit code 0
```

Alors, la ligne que nous avons écrit dans l'IDE `print("Hello World!")` est une instruction qui demande à l'ordinateur d'afficher `Hello World!` à l'écran. Nous pouvons écrire n'importe quoi à la place de `Hello World!` et l'ordinateur se chargera d'afficher le message que nous avons demandé.

Avant de continuer, nous allons détaillé ce que PyCharm nous a affiché :

/usr/bin/python3.5 c'est le chemin vers l'interpréteur python que PyCharm a utilisé pour exécuté notre programme. La majorité des programmes sous *Linux* sont installés dans le dossier `/usr/bin/` .

/home/no_one/PycharmProjects/hello_world/program.py ça c'est notre fichier (là où on est entrain d'écrire notre code source). Donc d'après cette ligne, on comprend que `python3.5` est entrain d'exécuter ce fichier.

Hello World! c'est le message que nous avons demandé à python d'affiché (on voit qu'il a bien obéi).

Process finished with exit code 0 ceci est un message affiché par `PyCharm` pour nous indiquer que notre programme s'est terminé avec succès.

Donc, `print()` est l'instruction responsable d'afficher un message à l'écran. On lui donne entre les `""` le message qu'on veut afficher. Essayez de Remplacer `Hello World!` par `Salut tout le monde` et ré-exécuté pour voir le nouveau résultat.

1.3.1 Les commentaires

Avant de terminer cette section avec un TP, nous allons parler des commentaires. Dans n'importe quel langage de programmation (y compris python), on a la possibilité d'écrire des commentaires.

Alors c'est quoi les commentaires ? Ce sont des bouts de textes qu'on écrit dans notre code source et qui permettrons d'expliquer le fonctionnement de notre code source. Ça peut vous semblez bizarre et inutile, mais sachez que c'est très commun qu'on revient à notre programme après quelques semaines pour l'améliorer ou pour le modifier, et sans les commentaires on sera obligé de tout repenser du début. Ne vous faites pas avoir par le fait que c'est vous qui a crée ce programme, car même si vous comprenez tous sur votre programme maintenant, vous oublierai après un certain temps.

Les commentaires ne serviront pas seulement à comprendre son code seulement, mais c'est très utile aussi quand vous travaillez en groupe pour crée un programme. Quand vous utilisez les commentaires, vous ne serrez pas obligé d'expliquer la partie que vous avez écrit à vos camarades.

J'insiste exprès sur le fait d'utiliser les commentaires, ils sont très très utiles et indispensables.

Alors, on sait ce que sont les commentaires, mais comment les utiliser avec `python` ? Voici un exemple sans plus tardé :

```
1 # Cette instruction affiche Hello World! a l'ecran.
2 print("Hello World!")
```

Les commentaires commence par un `#`, tous ce qui suit ce caractère est ignoré par `python`. Donc on peut écrire autant de ligne de commentaire qu'on veut, il faut juste précédé chaque ligne avec un `#`.

1.3.2 Exercice 1

Bon, on a assez bavardé, vous allez écrire votre vrai premier programme crée par vous même. Voici l'exercice :

- Créez un nouveau projet et nommé le "exercice 1".

- Créez un fichier dans ce projet où nous allons mettre notre code source. nommé ce fichier “main”.
- Débrouillez vous pour que le programme affiche :

```
Salut tout le monde.
On est dans une formation python, et on va cree des programmes super cool :D.
```

- N’oubliez pas d’utiliser les commentaires pour expliquer votre code.

1.3.3 Solution de l’exercice

```
1 # Ces deux instructions affiche des messages a l'ecran.
2 print("Salut tout le monde.")
3 print("On est dans une formation python, et on va cree des programmes super co
```

1.4 Les variables

Les variables sont une notion fondamentale en informatique et il sont utilisé dans tous les langages de programmation sans exception. Il ont plusieurs définitions, mais je pense que celle qu'on va voir est la plus simple.

En faite, votre ordinateur est une armoire géante qui contient des millions et des millions de tiroirs, et chaque tiroir est utilisé pour stocker des informations. Donc quand vous voulez stocké une information (disons votre age) pour l'utilisé plus tard dans votre programme, vous demandez à votre ordinateur de vous allouer un tiroir. Ce tiroir vous serra allouer jusqu'à la fin de votre programme.

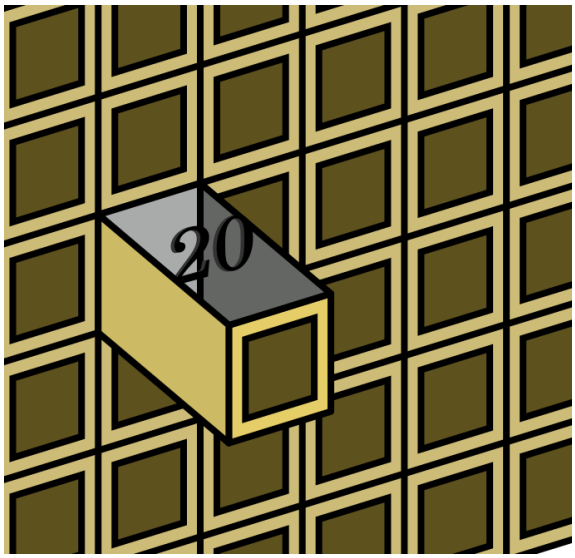


FIGURE 14 – Une tiroir qui stocke notre information

Ce tiroir est le votre, vous pouvez placez des information dedans, les consulter, les retirer.

Donc, où sont les variables dans tous ça ??

Les variables sont des noms qu'on associé aux tiroirs que nous avons demandé. Donc, dans l'exemple précédent, nous avons utilisé un tiroir pour stocker un age, donc nous allons nommé ce tiroir `age`. Et à chaque fois qu'on veut consulter ou modifier le contenu de ce tiroir, (supposons vous avez oublié votre age) on utilise le mot `age`.

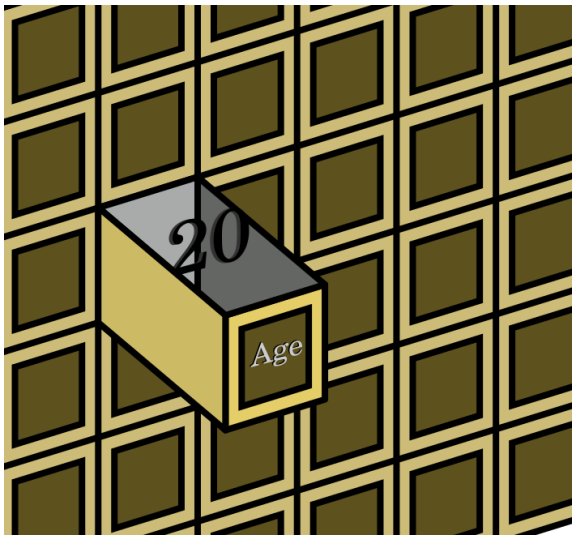


FIGURE 15 – Une tiroir qui stocke notre information, et qui est accessible par le nom age

1.4.1 Créer et utiliser les variables avec python

Pour créer une variable avec python, rien de plus simple. Il suffit d'écrire le nom de notre variable :

```
1 age = 20
```

Avec cette instruction, nous avons maintenant un tiroir réservé à nous, ce tiroir est accessible en écrivant le mot `age` dans python.

Si on veut voir le contenu de notre tiroir (variable), on peut utiliser l'instruction qu'on a vu précédemment `print()` (celle qui permet d'afficher à l'écran) :

```
1 age = 20
2 # Cette instruction va afficher mon age.
3 print(age)
```

On peut même modifier le contenu de notre variable à n'importe quel moment :

```
1 age = 20
2 # Cette instruction va afficher mon age.
3 print(age)
4
5 # Ces deux instructions vont me rendre plus vieux,
6 # et afficher mon nouveau age.
7 age = 30
8 print(age)
```

Voici ce qui nous serra affiché par PyCharm après l'exécution de notre programme :

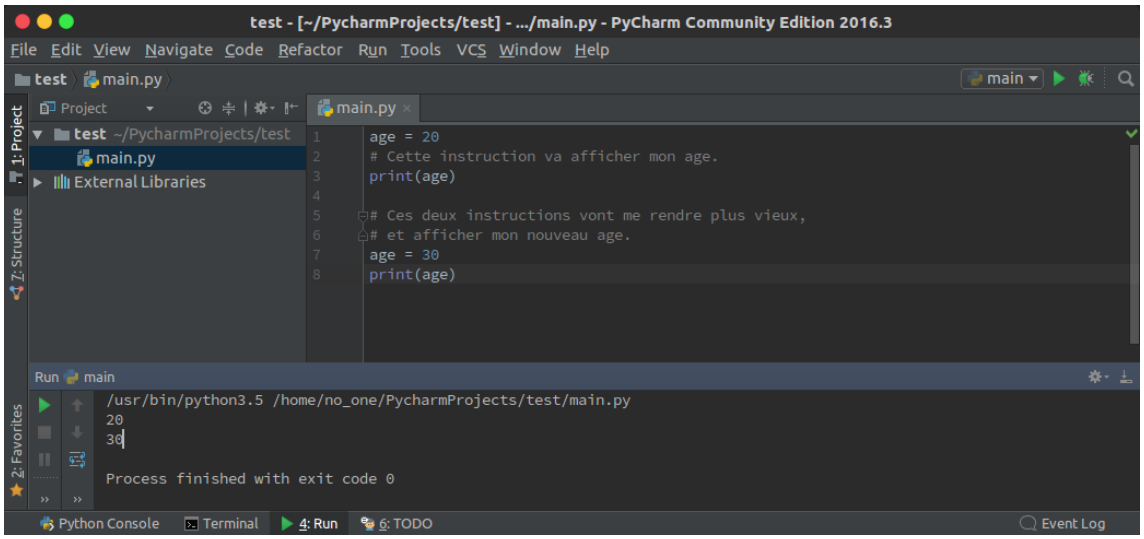


FIGURE 16 – Affichage de la variable age

1.4.2 Quel genre d'information on peut stocker dans nos variables

Il existe essentiellement 3 types d'information qu'on peut stocker dans les variables :

Les entiers : comme ce que nous avons stocké dans la variable `age` que nous avons utilisé jusque là. Les entiers sont appelé `int` en python.

Les valeurs décimales : des nombres à virgules, (comme : 3.453). Ces nombres sont appelé `float` en python. Exemple :

```
1 pi = 3.14
```

Du texte : on les appels les chaînes de caractères ou `str` en python. On les définit on les entourant par `""` ou `''`. Exemple :

```
1 prenom = "Amine"
2 prenom2 = "Moustapha"
3 nom = 'Malaoui'
```

1.4.3 Afficher les variables

Vous savez déjà qu'on peut afficher le contenu d'une variable avec `print()`, mais ce que vous ne savez pas c'est que `print()` permet d'afficher plusieurs variables à la fois. Il suffit juste de les séparer par des `,` :

```
1 # Je cree mes variables
2 prenom = "Amine"
3 nom = "Malaoui"
4
5 print("Salut, je m'appelle", prenom, nom)
```

En exécutant notre programme, PyCharm nous affichera ça :

```
/usr/bin/python3.5 /home/no_one/PycharmProjects/test/main.py
Salut, je m'appelle Amine Malaoui

Process finished with exit code 0
```

1.4.4 Récupérer une saisie

Ce que nous avons fait jusque là c'est de crée nos variables, et de les données des valeurs directement dans le code source. Ce que nous allons apprendre maintenant c'est de demander de l'utilisateur de donner une valeur à notre variable au cours d'exécution.

Pour cela on va utiliser l'instruction de récupération de saisie, qui est `input()` :

```
1 # On demande a l'utilisateur de nous donner son prenom,
2 # et on stock son prenom apres dans la variable prenom.
3 prenom = input("Veuillez entrer votre prenom : ")
4
5 # On dit bonjour a l'utilisateur.
6 print("Bonjour", prenom)
```

Ce que fait ce code c'est qu'il affiche à l'utilisateur `Veillez entrer votre prenom :`, et il lui donne la main après pour qu'il entre son prénom. Son prénom est maintenant stocker dans la variable `prenom`. Voila ce que PyCharm nous affiche après l'exécution :

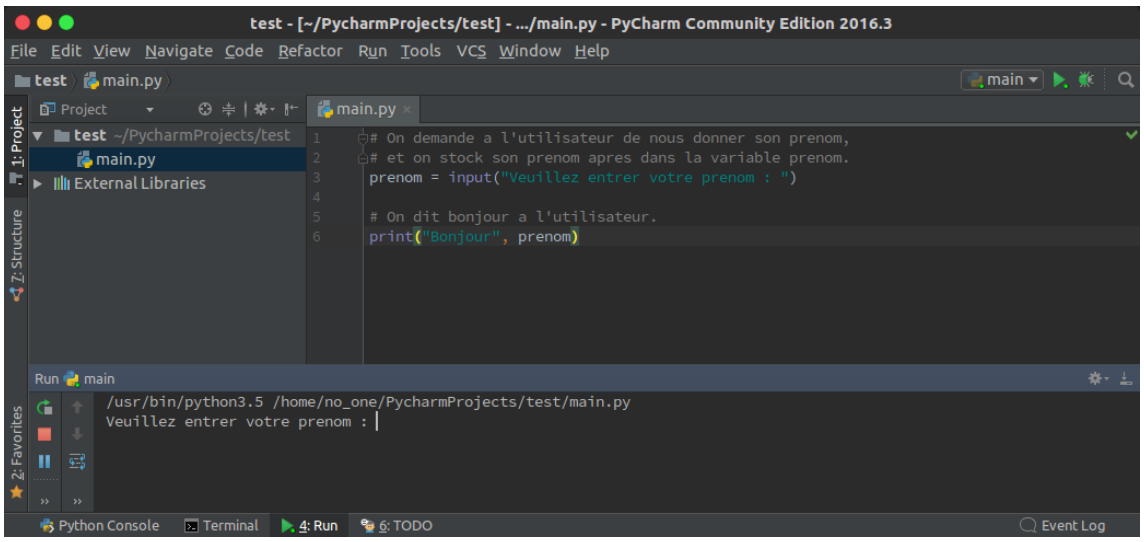


FIGURE 17 – PyCharm attend qu'on entre le prénom

Dès que l'utilisateur entre son nom et tape `Entrée`, PyCharm lui affiche `Bonjour` suivi du prénom de l'utilisateur :

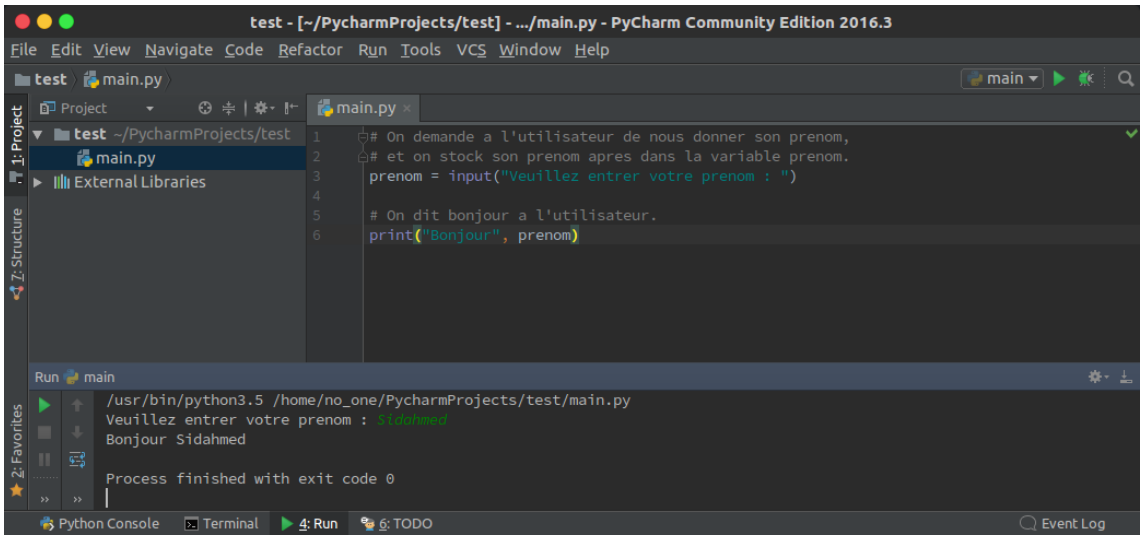


FIGURE 18 – PyCharm affiche Bonjour suivi du prénom

1.4.5 Comment nommé les variables

On va parler de la meilleur manière de nommé les variables avant de finir cette partie. vous avez juste quelques règles à suivre, et tout passera à merveille :

- Le nom d'une variable doit commencer par une lettre. Exemple :

```
# Faux
1prenom = 20

# Juste
prenom1 = 20

# Juste
prenom = 20
```

- Donner toujours des noms explicites à vos variables. C'est à dire en observant le nom de votre variable, il faut que vous serrez capable de comprendre son fonctionnement. Croyez moi, ça vous sera extrêmement utiles quand vous réexaminer votre code.
  Par exemple, si on veut stocker l'age d'un utilisateur :

```
# Faux, car c'est pas claire que x designe un age.
x = 20

# Juste
age = 20
```

- Si le nom de votre variable doit être composé de plusieurs mots, séparer les mots par un `_`. Exemple :

```
# Faux
notedemath = 15

```

```
# Juste
note_de_math = 15
```

- Et enfin, n'utilisez pas de caractères avec accents (comme `é`, `è`, `à` …) car ça peut vous posez des problèmes plus tard.

Enfin, nous avons terminé de parler sur les variables. Nous allons faire un petit exercices avant de passer à la section suivante.

1.4.6 Exercice 2

Ne sautez pas ces exercices, c'est très important de pratiquer en informatique.

Voici l'énoncé de l'exercice :

- Créez un nouveau projet et nommé le "exercice 2", et créez un fichier "main.py" dans ce projet.
- Débrouillez vous pour que le programme demande le "prenom", "nom" et "age" de l'utilisateur. Puis affichez tous ça dans une phrase du genre : `Tu t'appelles Yanis Khelloufi et tu as 22 ans.`.

1.5 les opérateurs arithmétique

Les ordinateurs sont des vrai machines à calcul, c'est ce qu'il savent faire le mieux. Donc dans cette partie nous allons apprendre à faire des calcules avec python.

Il y a dans python 5 symboles qui nous permettent de faire des opérations arithmétique :

- `+` fait l'addition.
- `-` fait la soustraction.
- `*` fait la multiplication.
- `/` fait la division entière.
- `//` fait la division euclidienne.
- `%` donne le reste de la division euclidienne (on appel ça le *modulo*).
- `**` calcul la puissance.

1.5.1 Addition, soustraction, multiplication

Pour l'addition et la soustraction et la multiplication, rien de plus simple, on utilise seulement les symboles approprié, et tout passe à merveille. Exemple :

```
1 x = 10 + 5
2 y = 21 - 7
3 z = 9 * 9
4
5 # Ceci affichera : 15 14 81
6 print(x, y, z)
```

1.5.2 Division entière et Division euclidienne

Pour voir la différence, un exemple vaut mieux qu'un long discours :

```
1 division_entiere = 10 / 3
2 division_euclidienne = 10 // 3
3
4 # Ceci affichera : 3.3333333333333335  3
5 print(division_entiere, division_euclidienne)
```

1.5.3 Modulo (reste de la division euclidienne)

Exemple :

```
x = 10 % 3

# Ceci nous affichera 1. Car le reste de la division de 10 par 3 donne 1.
print(x)
```

1.5.4 La puissance

Rien de plus simple. Exemple :

```
x = 5 ** 2

# Ceci affichera 125
print(x)
```

1.5.5 Opérations entre variables

Sachez bien que toutes les opérations que nous avons vu précédemment sont aussi applicables sur les variables. Donc on peut faire des calculs entres variables. Exemple :

```
a = 10
b = 15
resultat = a + b

# Ceci affichera : 10 + 15 = 25
print(a, "+", b, "=", resultat)
```

1.5.6 Bonus

Dans la majorité des autres langages de programmation, pour permuter deux variables (échanger leurs contenus), il faut toute une série d'opérations.

Exemple : disons on a les variables `a` et `b`, pour permuter leurs valeurs dans les autres langages de programmation, il faut faire ça :

```
# Il faut ces trois opérations pour permuter les valeurs
# de a et b
variable_temporaire = a
a = b
b = variable_temporaire
```

Alors en python, il suffit juste de faire :

```
# cette instruction permutent les valeurs de a et b
a, b = b, a
```

Vous pouvez créer les variables `a` et `b`, essayez ce code, et afficher ces deux variables pour voir le résultat.

1.5.7 Un petit problème !

Essayez ce code :

```
1
2 date_de_naissance = input("Veuillez entrer votre date de naissance : ")
3 age = 2017 - date_de_naissance
4
5 print("Votre avez", age, "ans")
```

Malheureusement, PyCharm n'a pas pu exécuter notre programme, et il nous affiche une erreur :

```
/usr/bin/python3.5 /home/no_one/PycharmProjects/test/main.py
Veuillez entrer votre date de naissance : 1994
Traceback (most recent call last):
  File "/home/no_one/PycharmProjects/test/main.py", line 3, in <module>
    age = 2017 - date_de_naissance
TypeError: unsupported operand type(s) for -: 'int' and 'str'

Process finished with exit code 1
```

Ce qui s'est passé c'est que l'instruction `input()` récupère la saisie sous forme de texte. Donc ce qu'elle a stocké dans la variable `date_de_naissance` n'est pas le nombre `1994` mais plutôt le texte (chaînes de caractères) `"1994"`. Donc quand Python arrive à l'instruction de la soustraction, il essaye de soustraire un texte d'un entier, et là ça plante, car ce n'est pas possible.

La solution consiste à convertir notre variable `date_de_naissance` en un entier. C'est extrêmement simple, il suffit juste d'utiliser l'instruction `int()`. Voici à quoi rassemblerai notre nouveau code source :

```
1 date_de_naissance = input("Veuillez entrer votre date de naissance : ")
2 # On converti notre variable en un entier.
3 date_de_naissance = int(date_de_naissance)
4
5 age = 2017 - date_de_naissance
6
7 print(age)
```

Et cette fois, le programme marche à merveille.

N'oubliez pas, dorénavant, quand vous utilisez `input()`, mettez en tête qu'elle récupère la saisie sous forme de texte. Pensez donc à convertir votre variable si vous voulez l'utiliser comme entier (utilisez `int()`) ou nombre à virgule (utilisez `float()`).

1.5.8 Exercice 3

Voici l'énoncé de l'exercice :

- Créez un nouveau projet et nommé le “exercice 3”, et créez un fichier “main.py” dans ce projet.
- Demandez l’age de l’utilisateur, et dite lui à quel année il aura 100 ans.

1.6 Les conditions

Les conditions sont des concepts fondamentaux de n'importe quel langage de programmation. Mais il sont un petit peu difficile à comprendre au début, mais vous verrez que dés que vous comprenez les conditions, vous serrai déjà capable de crée un programme disons intelligent et qui réagit selon la situation.

Par exemple : vous pouvez demander l'age de l'utilisateur, et puis lui affiche un message selon son age. Si il a plus de 18 ans, vous lui affichez `Vous êtes majeur`, sinon vous lui affichez `Vous êtes mineur`.

Avant de commencer de parler des conditions, il y a un ensemble de symbole que vous devez apprendre par cœur. Ces symboles sont utilisés pour réaliser les conditions :

Symbole	Signification
==	est égal à
>	est supérieur à
<	est inférieur à
>=	est supérieur ou égal à
<=	est inférieur ou égal à
!=	est différent de

TABLE 1 – Symboles de comparaison

1.6.1 Le if

Pour réaliser une condition minimal en python, il faut juste suivre un schéma simple, le voici :

```
1 if condition :
2     instruction1
3     instruction2
4     .
5     .
6     .
7     instructionN
```

Donc, la condition `condition` sera évalué par python, si elle est vrai, il exécute toutes les instruction qui se trouvent dans le bloc du `if`, et si elle est fausse, alors il va sauter toutes les instructions qui se trouvent dans le bloc du `if`.

Attention, il faut que les instructions soient décalé par rapport à notre `if` comme le montre l'exemple précédent. C'est comme ça que python comprend quelles sont les instruction qui appartiennent au `if` et quelles sont les instructions qui appartient au programme principale.

Juste pour titre informatif, ce décalage est appelé *Indentation*, et l'ensemble d'instruction qui se trouvent dans le `if` est appelé *Un bloc de code*.

On donne un exemple :

```
1  age = 20
2
3  # On test si l'age est superieur a 18 pour afficher un message.
4  if age > 18 :
5      print("Vous etes majeur")
6
7  print("Au revoir")
```

Si on exécute ce code, on voit que le message `Vous etes majeur` est affiché, et même le message `Au revoir` est affiché. Mais si on modifie l'age pour lui donner une valeur inférieur à `18`, alors seulement le message `Au revoir` est affiché, car cette instruction n'est pas inclut dans le bloc du `if`.

1.6.2 Le else

Pour enrichir encore plus notre code, on peut utilisé le `else`. Ça permet de désigner un bloc sinon, ce bloc sera exécuté si la condition évalué dans le `if` est fausse. Un exemple :

```
1  age = 20
2
3  # Si l'age est supérieur a 18 on affiche : Vous êtes majeur.
4  # Sinon (c'est a dire l'age inférieur ou égale a 18) alors
5  # on affiche : Vous êtes mineur.
6  if age > 18 :
7      print("Vous êtes majeur")
8  else :
9      print("Vous êtes mineur")
10
11 print("Au revoir")
```

Si on modifie l'age pour qu'il soit inférieur ou égale à `18`, on remarquera que cette fois c'est le message `Vous etes mineur` qui sera affiché. Le message `Au revoir` est toujours affiché, car il n'appartient ni au `if` ni au `else`.

1.6.3 Le elif

Le `elif` est une abréviation de `else if` qui veut dire *Sinon Si*. Il se situe entre le `if` et le `else`, et ça permet de tester une autre condition.

Ça peut vous semblez abstrait, mais un exemple va éclaircir le concept :

```
age = 20

# Si l'age est superieur a 70 alors on affiche : Salut papi
# Sinon si l'age est superieur a 40 alors on affiche : Salut monsieur
# Sinon si l'age est superieur a 18 alors on affiche : Salut jeune
# Sinon on affiche le message qui se trouve dans le else.
if age > 70 :
    print("Salut papi")
elif age > 40 :
    print("Salut monsieur")
elif age > 18 :
    print("Salut jeune")
else :
    print("Tu n'appartient à aucune catégorie, dégage -_-")

# Ce message est toujours affiché.
print("Au revoir")
```

En utilisant le `elif`, on pourra re-tester la valeur de la variable age une autre fois. Amusez vous à changer la valeur de la variable `age`, et vous verrez qu'à chaque fois l'un des 4 messages est affiché.

Faites très attention, un seule bloc d'instruction est exécuté parmi tous ces 4 blocs. Donc si on donne la valeur `70` à notre variable `age`, seul bloc de code du `if` sera exécuté, et tout les autres `elif` et le `else` seront ignoré.

1.6.4 Comparez les chaînes de caractères

Sachez que c'est testes qu'on vient de voir ne sont pas applicables seulement au nombres, on peut aussi les appliquer au chaînes de caractères. Exemples :

```
mot_de_passe = input("Veuillez entrer votre mot de passe: ")

if mot_de_passe == "MonSuperMotDePasse" :
    print("Bienvenu")
else
    print("Dégaaaaaaaaaage imposteur")
```

1.6.5 Plusieurs conditions à la fois

Jusque là, nous avons tester une seule condition dans le `if` ou le `elif`, par exemple "Si l'age est supérieur à 18". Mais comment faire si on veut tester "Si l'age est supérieur à 18 et inférieur à 40 au même temps"?? Pour cela, il suffit d'utiliser les opérateurs logique. Ces opérateurs permettent de combiner plusieurs condition à la fois.

Il y a un ensemble de règles à apprendre pour réaliser cela :

Symbole	Signification
and	ET
or	OU
not	NON

TABLE 2 – Les opérateurs logiques

Voici quelques exemples du fonctionnement de ces opérateurs :

```
# Si l'age est supérieur à 18 et inférieur à 40 au même temps.
if age >= 18 and age <= 40:
    # Les instructions
```

```
# Si on est Samedi ou Vendredi alors on entre dans le bloc du if.
if jour == "Samedi" or jour == "Vendredi" :
    # Pas d'études
```

```
# Le 'not' est utilisé pour exprimé la négation d'une condition.
# Ça se traduit en : si l'age n'est pas supérieur à 18.
if not age > 18 :
    # Les instructions
```

Sachez qu'on peut combinez les opérateurs logiques entres eux, mais pour cela vaut mieux utiliser les parenthèses pour exprimer la priorité. Exemple :

```
if (jour == "Samedi" or jour == "Vendredi") and chaleur < 35 :
    # Sortir de la maison
```

1.6.6 Les booléens

Il y a un autre type de variables dont je ne vous ai pas parlé avant pour ne pas vous en charger. Ce type de variable est appelé *Booléen* (`bool` en python) et il est très utilisé dans les conditions et les boucles.

Les booléens sont des variables qui peuvent contenir une de deux valeurs seulement : `True` (qui veut dire vrai) ou `False` (qui veut dire faux), et on peut testé directement si ces variables sont à vrai dans les conditions :

```
heureux = True

# Si la variable heureux est vrai, alors on entre dans le if,
# Sinon on entre dans le else.
if heureux :
    print("Ah, je vois que t'es heureux aujourd'hui")
else :
    print("Mais pourquoi t'es triste ??")
```

Les opérateurs logiques sont applicable sur les booléens. Exemple :

```
# Ça se traduit en : Si la personne et heureuse et
# le jour et beau alors Sorte de la maison.
if heureux and beau_jour :
    # Sortir de la maison
```

1.6.7 Exercice 4

Voici l'énoncé de l'exercice :

- Créer un projet nommé le 'exercice 4' et créer un fichier 'main.py' dedans.
- Demandez un nombre de l'utilisateur, et dites lui si ce nombre est paire ou impaire.

1.7 Les boucles

Si vous avez compris les conditions, les boucles seront un jeu d'enfant. En gros, les boucles servent à répéter certaines instructions plusieurs fois, et on a deux boucles dans le langage python :

- La boucle `while` qui signifie *tant que*.
- La boucle `for` qui signifie *pour*.

1.7.1 La boucle while

La boucle `while` permet de répéter un bloc d'instructions *tant que* une condition est vrai. Exemple :

```
1 touche = ""
2
3 while touche != "q" :
4     touche = input("Tapez q pour sortir : ")
5
6 print("Au revoir")
```

Dans cet exemple, on demande à l'utilisateur d'entrer la lettre `q`. Si il reste sage et il fait ce que nous le avons demandé, alors on lui laisse sortir de la boucle, sinon on lui redemande à nouveau jusqu'à ce qu'il entre la touche `q`.

On peut utilisé les booléens avec la boucle `while`. Exemple :

```
1 heureux = True
2
3 while heureux :
4     # Continuez de sourire.
```

On donne un autre exemple de l'utilisation de la boucle `while`. Imaginez que votre prof vous demande d'écrire un programme qui affiche le message "Ceci est un message très utile" 10 fois, vous pouvez faire ça :

```
1  print("Ceci est un message très utile")
2  print("Ceci est un message très utile")
3  print("Ceci est un message très utile")
4  print("Ceci est un message très utile")
5  print("Ceci est un message très utile")
6  print("Ceci est un message très utile")
7  print("Ceci est un message très utile")
8  print("Ceci est un message très utile")
9  print("Ceci est un message très utile")
10 print("Ceci est un message très utile")
```

C'est vrai que ça accomplira ce que votre prof a demandé, mais ce n'est ni propre ni pratique. Et imaginez encore si cette fois elle vous demande d'afficher ce message 100 fois, ou 1000 fois …on voit bien que cette technique n'est plus efficace.

Ce qu'on peut faire pour réaliser ce que le prof a demandé c'est d'utiliser la boucle `while` et un *compteur* :

```
1 # On initialise notre variable qui nous servira de compteur.
2 i = 0
3
4 # Tant que la variable compteur est inférieur à 10 on
5 # affiche un autre message.
6 while i < 10:
7     print("Ceci est un message très utile")
8     # On incrémente notre compteur pour savoir quand s'arrêter.
9     i = i + 1
```

Voici le résultat d'exécution :

```
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
Ceci est un message très utile
```

La variable `i` est souvent utilisé en informatique comme un compteur ou pour parcourir les tableaux (on verra ça plus tard).

En informatique, dans la majorité des cas, on commence nos compteur de `0` et pas de `1`. Ça a une relation avec le parcours des tableaux qu'on verra dans les prochaines sections. Donc si vous voulez utilisé un compteur, il faut de préférence commencer de `0`.

1.7.2 La boucle for

La boucle `for` est utilisé uniquement avec les compteurs. C'est une forme compacté de la boucle `while` précédente. Si on refait l'exemple précédent en utilisant la boucle `for` ça nous donne ça :

```
1 # Ça se traduit en : pour i allant de 0 à 10 (exclus).
```

```
2
3 for i in range(0, 10):
      print("Ceci est un message très utile")
```

Dans cet exemple, `i` prend chacune des valeurs `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`. Et à chaque fois on affiche le message `Ceci est un message très utile`.

Pour voir que `i` prend vraiment toute ces valeurs là, on peut afficher la variable `i` à chaque itération de la boucle :

```
1 for i in range(0, 10) :
2     print(i)
```

Ce qui nous donne ça :

```
0
1
2
3
4
5
6
7
8
9
```

On a vu la boucle `for`. Maintenant détaillons un peu comment l'utiliser :

```
1 for i in range(depart, arrivee) :
2     print(i)
```

La variable `i` parcourra toute les valeurs en démarrant de la valeur `depart` (inclus) jusqu'à la valeur `arrivee` (exclus). Amusez vous à changer la valeur de `depart` et `arrivee` pour voir quelles sont les valeurs pris par `i`.

1.7.3 Un bonus de l'utilisation de la boucle for

Dans certaines situations, on utilise la boucle `for` de cette manière :

```
1 for i in range(depart, arrivee, pas) :
2     print(i)
```

C'est comme l'utilisation précédente, mais là on a ajouté le `pas`. Ce `pas` est utilisé pour dire à la variable `i` combien elle doit sauter à chaque itération de la boucle. Essayez cette exemple et vous verrez :

```
1 for i in range(0, 10, 2) :
2     print(i)
```

L'exécution nous donne ça :

```
0
2
4
6
8
```

`i` démarre de `0`, mais à chaque itération de la boucle, elle fait un saut de `2`.

1.7.4 Exercice 5

Voici l'énoncé :

- Créez un projet nommé 'exercice 5' et créez un fichier dedans nommé 'main.py'.
- Demandez à l'utilisateur d'entrer un nombre, et affichez lui tous les diviseurs de ce nombre. Exemple : S'il entre `10`, vous lui affichez `1`, `2`, `5`, `10`.

1.8 Découper le programme en fonctions

Jusque là, nous avons écrit toutes nos instructions directement , l'une après l'autre. Maintenant, nous allons apprendre à découper nos programmes en petit bouts, et cela grâce aux fonctions.

1.8.1 C'est quoi une fonction ?

Une fonction est un ensemble d'instructions qu'on groupe ensemble. Cet ensemble d'instructions exécute des actions et renvoient un résultat.

Il vaut mieux que vous comprenez les fonctions avec les exemples, alors on va donner plusieurs exemples et on va les expliquer un à un.

1.8.2 Fonction simple

Voici l'exemple d'une fonction qui calcule le triple d'un nombre :

```
1 def triple(nombre) :
2     resultat = nombre * 3
3     return resultat
```

Et voici comment on va utiliser cette fonction dans notre programme :

```
1 # Instructions ici
2 x = triple(7)
3
4 # Ceci nous affichera 21
5 print(x)
```

Des explications s'impose ici :

- `def triple(nombre) :` : `def` est un mot clé utilisé par python, ce mot nous permet de crée les fonctions. `triple` est le nom de notre fonction, c'est ce mot qui nous permet d'utiliser notre fonction dans notre programme principale. `nombre` est une variable qui est crée à l'intérieur de la fonction, cette variable prend la valeur qu'on donne à notre fonction au moment de l'utiliser (dans cette exemple elle prend la valeur `7` à cause de l'instruction `x = triple(7)`); une fois la fonction est fini, la variable `nombre` est effacé.
- `resultat = nombre * 3` : ceci est une instruction qui est exécuté dans notre fonction. On peut mettre autant d'instruction qu'on veut dans la fonction, il faut juste ne pas oublier de les décaler (comme on a vu dans les conditions).
- `return resultat` : cette ligne renvoie la valeur de la variable `resultat` au programme principale.

- `x = triple(7)` : cette instruction appel la fonction `triple()` en lui donnant la valeur `7`. Ensuite, la valeur renvoyer par la fonction `triple()` sera stocker dans la variable `x`.

Donc voici notre programme en tout :

```
1 def triple(nombre) :
2     resultat = nombre * 3
3     return resultat
4
5
6 # C'est d'ici que commence l'exécution de notre programme.
7 x = triple(7)
8
9 print(x)
```

N'oubliez pas de décaler les instructions dans la fonction, sinon python ne fera pas la différence entre les instructions qui sont dans la fonctions et les instructions qui sont dans le programme principale.

1.8.3 Plusieurs paramètres

On peut utiliser plusieurs variables entre les parenthèses de la fonction, et pas seulement une comme on a fait dans l'exemple précédent. Par exemple, on a ici une fonction qui calcule la somme de deux nombres :

```
1 def addition(nombre1, nombre2) :
2     resultat = nombre1 + nombre2
3     return resultat
```

Les variables qui sont créent entre les parenthèse de la fonction sont appelé *paramètres*. On les appels comme ça dans le sens où se sont des paramètres qu'on passe à la fonction pour qu'elle fait ces calculs.

Et voici notre programme qui utilise cette fonction :

```
1  def addition(nombre1, nombre2) :
2      resultat = nombre1 + nombre2
3      return resultat
4
5  # C'est ici que commence l'exécution de notre programme.
6  a = 10
7  b = 25
8
9  x = addition(a, b)
10
```

```
print(x)
```

1.8.4 Fonction qui ne renvoie rien

On peut aussi crée une fonction qui ne renvoie rien. Exemple :

```
def presentation(nom, prenom, age) :
    print("Bonjour.")
    print("Je m'appelle", nom, prenom, "et j'ai", age, "ans")
```

Et voici le programme qui l'utiliser :

```
def presentation(nom, prenom, age) :
    print("Bonjour.")
    print("Je m'appelle", nom, prenom, "et j'ai", age, "ans")

# C'est ici que commence l'exécution de notre programme.
presentation("Yanis", "Khelloufi", 22)
```

1.8.5 Fonction qui renvoie plusieurs valeurs

On peut aussi crée une fonction que renvoie plusieurs valeurs. Voici un exemple d'une fonction qui fait la division euclidienne, et renvoie le résultat de la division et le reste de la division :

```
def division_euclidienne(divisee, diviseur) :
    resultat_de_division = divisee // diviseur
    reste_de_division = divisee \% diviseur

    return resultat_de_division, reste_de_division
```

Pour récuperer ce que la fonction a renvoyé on fait ça :

```
# 'a' recevra la valeur de la variable 'resultat_de_division',
# et 'b' recevra la valeur de la variable 'reste_de_division'.
a, b = division_euclidienne(20, 3)
```

Voici le code complet de notre programme :

```
def division_euclidienne(divisee, diviseur) :
    resultat_de_division = divisee // diviseur
    reste_de_division = divisee \% diviseur

    return resultat_de_division, reste_de_division

# C'est ici que commence l'exécution de notre programme.
a, b = division_euclidienne(20, 3)

# Ceci nous affichera : 6 2
print(a, b)
```

1.8.6 Pourquoi découper notre programme ??

C'est vrai que les exemples que nous avons donnée jusque là ne semblent pas vraiment utiles. Mais sachez que se sont que des exemples pour que vous compreniez comment crée les fonctions. Quand on découpe vraiment, on découpe en des fonctions qui sont vraiment utiles, exemple : une fonction qui nous dit si un nombre est premier ou pas.

On revient à notre question, pourquoi découper ??

Et bien, pour faciliter la réalisation d'un programme et pour rendre notre code source plus lisible et plus organisé. Car les programmes que nous avons réalisé jusque là sont des programmes simples, qui ne dépassent pas une dizaine de lignes de code. Mais les vrai programmes sont beaucoup plus complexes que ça, et il peuvent atteindre les dizaines de milliers de lignes de codes très facilement. Sans découper, vous ne pouvez pas créez ou maintenir ce genre de programme.

1.8.7 Exercice 6

Voici l'énoncé de l'exercice :

- Créez un projet nommé 'exercice 6' et créez un fichier dedans nommé 'main.py'.
- Créez une fonction qui reçoit un nombre en paramètre, et qui nous dit s'il est premier où pas.

Cette fonction retourne `True` où `False`

Un nombre premier est un nombre qui est divisible seulement par `1` et par lui même. En d'autre terme, aucun autre nombre ne divise ce nombre

- Dans votre programme principale, demander un nombre de l'utilisateur, et dite lui si ce nombre est premier ou pas.