Premiers pas en Python

Introduction

Dans ce chapitre, on introduit le langage qui a été choisi pour l'informatique en classe préparatoire : Python. On l'utilise pour revoir des éléments d'algorithmique du lycée avec quelques éléments nouveaux. Tout ceci sera repris et approfondi ensuite.

I Le langage Python

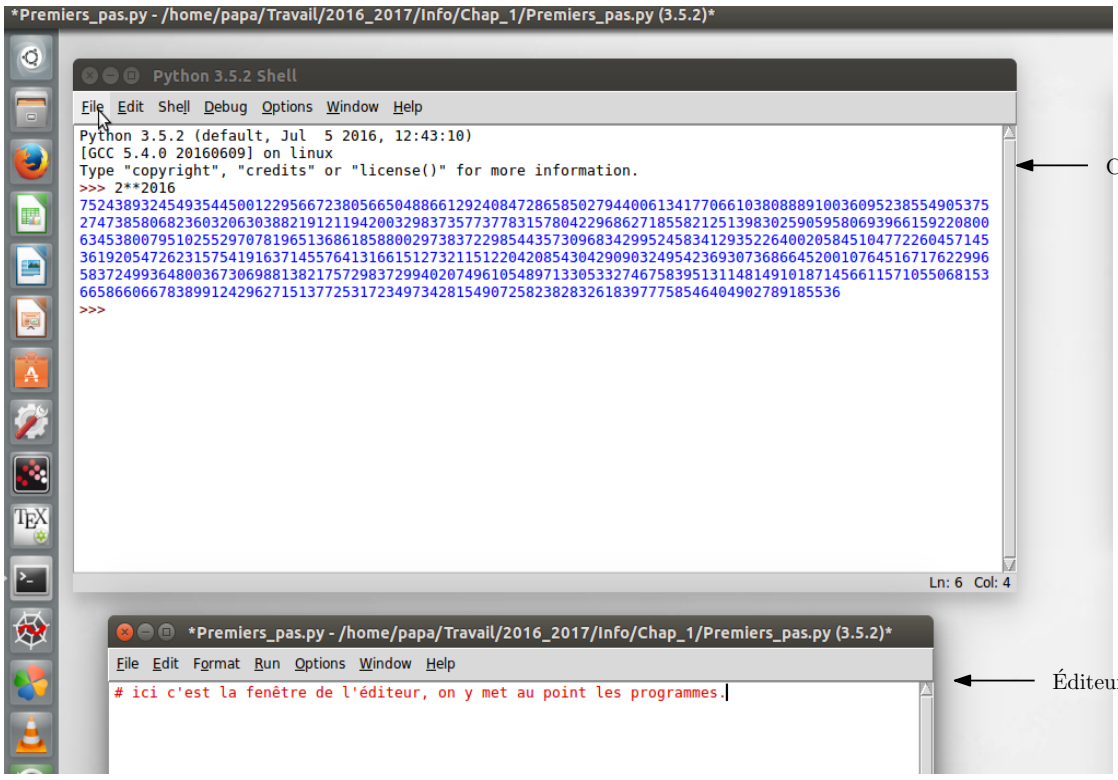
Définition (Langage de programmation)

> Un **programme informatique** est un texte qui décrit les opérations que l'on souhaite faire exécuter par un ordinateur. Ce texte est écrit en suivant les règles d'un **langage de programmation**. Dans ce cours, nous utiliserons le langage **Python** (version 3).

Un langage de programmation fait interface entre l'homme et la machine, il est compréhensible par le premier et exécutable par la seconde.

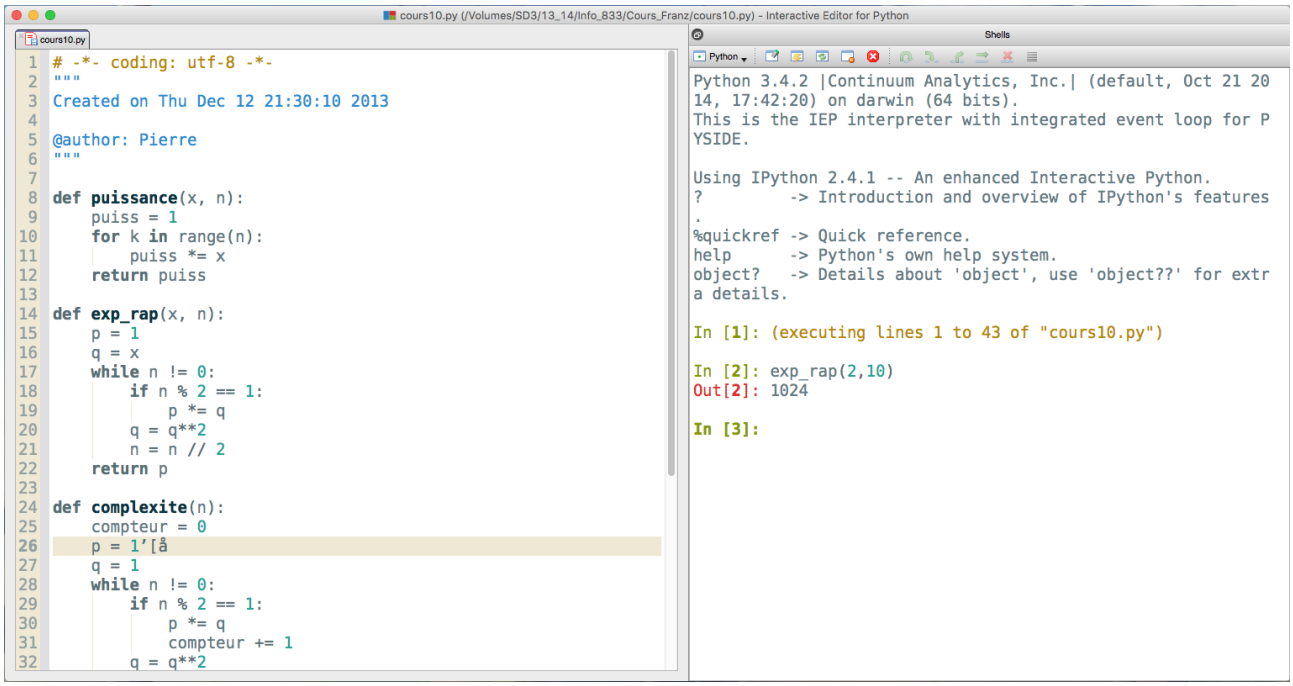
Pour rédiger et mettre au point un programme on utilise généralement un Environnement de Développement Intégré : EDI (ou IDE en anglais). Pour Python il y en existe de nombreux. On n'en présente ici que deux :

IDLE a l'avantage de la simplicité et il est généralement fourni par défaut avec l'installation standard du langage Python.



Lorsque l'on lance IDLE, une fenêtre de console (Python Shell) apparait. Elle sert à exécuter des commandes simples et aide à la mise au point des programmes. C'est aussi dans la console que l'on voit le résultat de l'exécution des programmes. Elle ne permet pas d'écrire du code. Vérifier la version de Python sur la première ligne du texte de la fenêtre. À l'aide du menu *File* de la console, on peut ouvrir une nouvelle fenêtre pour écrire un programme : C'est l'éditeur.

Pyzo est l'environnement mis à disposition des candidats lors de l'oral de maths 2 de Centrale. On a une unique fenêtre avec un cadre pour l'éditeur à gauche et un autre pour la console à droite [1].



I.1 Les calculs

```
>>> 2**100
1267650600228229401496703205376
```
1

```
>>> cos(1)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    cos(1)
NameError: name 'cos' is not defined
```
2

1. Il faudra désactiver les outils inutiles dans le menu outils si nécessaires.

Les fonctions usuelles des mathématiques doivent être importées du module math : une seule (ou plusieurs),

```
>>> from math import cos
>>> cos(1)
0.5403023058681398
```
3

ou tout le module.

```
>>> from math import *
>>> log(exp(2))
2.0
```
4

Un ordinateur, c'est une machine donc ça ne fait jamais d'erreur :

```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.551115123125783e-17
```
5

I.2 Les suites

I.2.1 Suites données par leur terme général

```
>>> u = (n + 1)/(5 - 2*n)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    u = (n + 1)/(5 - 2*n)
NameError: name 'n' is not defined
```
6

Oups !

Deuxieme essai :

```
>>> n = 1
>>> u = (n + 1)/(5 - 2*n)
>>> u
0.6666666666666666
```
7

La bonne façon de faire est de définir une fonction :

```
>>> def u(n):
        return (n + 1)/(5 - 2*n)

>>> u(100)
-0.517948717948718
```
8

Entraînement 1 

Écrire une fonction qui calcule le terme général de la suite $(u_n)_{n\geqslant 1}$ définie par $u_n = \dfrac{\ln(n)}{n}$.

I.2.2 Suites définies par récurrence

```
>>> def u(n):
        x = 1
        for k in range(1, n + 1): # k prends les valeurs de 1 à n
            x = 3*x - 1
        return x
```

9

- Cas particulier des **sommes** :

```
>>> def S(n):
        S = 0
        for k in range(1, n + 1):
            S = S + k
        return S
```

10

Que calcule la fonction F ?

```
>>> def F(n):
        p = 1
        for k in range(1, n + 1):
            p = p*k
        return p
```

11

Entraînement 2 

Écrire une fonction prenant un entier n en paramètre et qui renvoie la somme des carrés des n premiers entiers.

Entraînement 3 

Écrire une fonction prenant un entier n en paramètre et qui renvoie le produit des n premiers entiers impairs.

I.3 Détermination d'un seuil pour une limite

Dans la suite on présente un code comme il apparaitrait dans l'éditeur. On se limite ici à des cas où les suites sont monotones croissantes et divergent vers $+\infty$. Il s'agit de trouver le premier rang n à partir duquel la suite a ses termes plus grand qu'un nombre A donné.

- Cas d'une suite donnée par son **terme général** : $u_n = \dfrac{2n+1}{\sqrt{3+n}}$

```
from math import sqrt

def seuil(A):
    n = 0
    while (2*n + 1)/sqrt(3 + n) < A:
        n = n + 1
    return n
```

12

- cas d'une suite définie par **récurrence** :

```
def seuil(A):
    n = 0
    x = 1
    while x < A:
        n = n + 1
        x = x + 1/(x + 1)
    return n
```

13

Entraînement 4 

Écrire une fonction prenant en paramètre un nombre A et retournant la première valeur de n tel que : $1 + 2 + 3 + \cdots + n > \text{A}$. Ici, on demande de ne pas utiliser la formule qui donne la somme.

I.4 Les tests

```
from math import sqrt

def racines_trinome(a, b, c):
    delta = b**2 - 4*a*c
    if delta > 0:
        x1 = (-sqrt(delta) - b)/(2*a)
        x2 = (sqrt(delta) - b)/(2*a)
        return x1, x2
    elif delta == 0:
        return -b/(2*a)
    else:
        return None
```

14

Entraînement 5 

Écrire une fonction `valeur_absolue` qui prend un nombre x en argument et renvoie la valeur absolue de x. On n'utilisera pas la fonction `abs` de Python.

I.5 Approximation des zéros des fonctions

- La méthode du balayage

```
def balayage(f, x0, p):
    y0 = f(x0)
    y1 = f(x0 + p)
    while y0*y1 > 0:
        x0 = x0 + p
        y0 = y1
        y1 = f(x0 + p)
    return x0, x0 + p
```

15

- La dichotomie

```
def dichotomie(f, x0, x1, p):
    while x1 - x0 > p:
        m = (x0 + x1)/2
        if f(x0)*f(m) > 0:
            x0 = m
        else:
            x1 = m
    return x0, x1
```

16

I.6 Simulation en probabilité

La fonction random du module random renvoi un nombre flottant choisi de manière pseudo-aléatoire dans l'intervalle $[0, 1]$. On peut l'utiliser pour simuler une variable aléatoire binomiale :

```
from random import random

def binom(n, p):
    b = 0
    for k in range(1, n + 1):
        if random() < p:
            b = b + 1
    return b
```

17

Entraînement 6

On place un insecte sur un axe gradué. À chaque seconde il saute d'une unité vers la droite avec une probabilité p et vers la gauche avec une probabilité 1-p. Il effectue n sauts en tout.

Écrire une fonction marche_hasard(n, p) qui prend n et p comme paramètres et renvoie une simulation de la variable aléatoire qui donne la position finale de l'insecte.

I.7 Les listes

On peut parcourir bien autre chose que des séquences d'entiers consécutifs avec une boucle :

```
palette = ['bleu', 'vert', 'rouge', 'jaune']
for couleur in palette:
    print(couleur)
```
18

```
bleu
vert
rouge
jaune
```

Le premier élément d'une liste L est obtenu par L[0], le second par L[1], etc. La longueur est obtenue par **len**(L). La liste vide est notée [].

```
>>> palette.append('blanc')
>>> print(palette)
['bleu', 'vert', 'rouge', 'jaune', 'blanc']
```
19

II Ressources en ligne et installation de Python

Quelques liens vers des ressources :
- Site officiel : http ://www.python.org/ http ://www.python.org/download/ http ://www.python.org/doc/
- Cours sur Python : http ://inforef.be/swi/python.htm ou http ://python.developpez.com/cours/
- Tutoriel sur Python : http ://www.siteduzero.com/tutoriel-3-223267-apprenez-a-programmer-en-python.html
- Le site de l'Olympiade française d'informatique (France-ioi) : http ://www.france-ioi.org/algo/chapters.php.

Python est installé par défaut sur la distribution Linux Ubuntu, sous Windows ou MAC OSX on peut télécharger les installateurs à partir de http ://www.python.org/download/ et on disposera d'un environnement graphique Idle avec console et éditeur de texte.

Memento Python

1. **Variables, affectation**

   Pour stocker une valeur dans une variable, la syntaxe générale est :

   `nom_variable = expression`

   L'expression est évaluée et le résultat est ensuite stocké dans la variable.

   Exemples :

```
compteur = 0
```

   ou encore

```
compteur = compteur + 1
```

2. **Tests**

   Pour effectuer des tests, on utilise la syntaxe :

```
if Condition:
    instruction Bloc1
    instruction Bloc1
    ...
else:
   instruction Bloc2
   instruction Bloc2
   ....
```

   Lorsque l'on a besoin d'envisager plusieurs alternatives on intercale des `elif`.

3. **Boucles conditionnelles**

   Quand on veut répéter un certain nombre de fois une action et que le nombre n de répétitions est connu à l'avance on utilise une boucle inconditionnelle dont la syntaxe est

```
for k in range(n):
    instruction du bloc
    instruction du bloc
    ...
```

   Attention, lors des itérations de la boucle la variable entière `k` prend les valeur de 0 à $n-1$ et `range(4, 10)` permet de parcourir les entiers de 4 à 9.

4. **Boucles inconditionnelles**

   Lorsque l'on ne sait pas à l'avance combien de fois la boucle devra être exécutée, on utilise une boucle **while**. La syntaxe est la suivante :

```
while condition:
    instruction du bloc
    instruction du bloc
    ...
```

5. **Fonctions**

   Un programme doit toujours est conçue de manière **modulaire** pour cela on dispose des fonctions. La syntaxe générale pour définir une fonction est :

```
def nomFonction(param1, param2, ...):
    instruction
    instruction          # bloc d'instructions
    instruction
    ...
```

   Dans une fonction, l'instruction `return expression` provoque la sortie de la fonction est la valeur de l'expression est la valeur retournée par la fonction lors de son appel. Ne pas utiliser `print` pour renvoyer la valeur!!

6. **Listes**

   Les listes permettent de former des collections d'objets qui sont numérotés en commençant par 0. Elles sont délimitées par `[` et `]`.

   — La liste vide s'obtient par `L = []`.
   — Pour une liste `L[k]` renvoie l'élément d'indice `k` si il existe.
   — la méthode `.append` permet d'ajouter un élément à la fin de la liste.
   — Une liste est itérable avec l'instruction `for`.

```
>>> L = []
>>> L[0]
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    L[0]
IndexError: list index out of range
>>> L.append(8)
>>> L[0]
8
>>> L=[8, 3, 2]
>>> for k in L:
        print(k)


8
3
2
```