Ce qui a été vu au tp n°1

L'objectif était de réaliser un programme permettant de déterminer si un nombre entier est un nombre premier ou non.

Première approche : une approche brutale

Un nombre entier est premier si il n'admet que 2 diviseurs différents : 1 et lui-même.

15 n'est pas premier puisqu'il est divisible par 1, 3, 5 et 15.

17 est premier parce qu'il n'est divisible que par 1 et 17.

1 n'est pas premier parce qu'il n'est divisible que par 1.

Notion de doctring en Python

Une docstring en python indique ce que doit faire une fonction. Elle permet une meilleure compréhension du code mis en place pour des corrections ou des améliorations.

```
def est_premier(nombre):
    '''
    nombre : int
    déterminer si un entier naturel est premier
    return : True ou False (bool)
    >>>est_premier(15)
    False
    >>>est_premier(17)
    True
    >>>est_premier(1)
    False
    '''
```

On précise dans une docstring le type des variables utilisées :

- int pour integer : entier naturel,
- bool pour booléens : True ou False,
- float pour flottant : nombres décimaux.

  On peut utiliser la commande `type` pour déterminer le type d'un objet en python :

```
>>> type(12)
<class 'int'>
>>> type(12.5)
<class 'float'>
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Pour déterminer si un nombre n est premier, on va tester tous les nombres de 2 à n-1 pour déterminer si ce sont des diviseurs ou non de n :

- si le nombre est un diviseur, alors n n'est pas premier et on renvoie False,
- si on a parcouru tous les nombres de 2 à n-1 sans trouver de diviseurs, alors le nombre n est premier, on renvoit True

Proposition de programme :

```
def est_premier(nombre):
    '''
    nombre : int
    déterminer si un entier naturel est premier
    return : True ou False (bool)
    >>>est_premier(15)
    False
    >>>est_premier(17)
    True
    >>>est_premier(1)
    False
    '''
    # on utilise une boucle for pour parcourir tous les nombres
    # de 2 à nombre-1
    for diviseur in range(2,nombre) :
        # on calcule le reste de la division euclidienne
        # du nombre par le potentiel diviseur
        if nombre%diviseur==0: # si le reste est égal à 0
            return False # le nombre n'est pas premier, on renvoit False
    # si on n'a pas trouvé de diviseur en parcourant la boucle
    return True # le nombre est premier, on renvoit False
```

boucles for en python :

Considérons la fonction suivante :

```
def boucle(n):
    for indice in range(2,n): # indice prend les valeurs entières de 2 à n-1
        print(indice)
```

Après enregistrement et exécution dans la console, on a les résultats suivants :

```
>>> boucle(5) # affiche les valeurs entières de 2 à 5-1=4
2
3
4
>>> boucle(7) # affiche les valeurs entières de 2 à 7-1=6
2
3
4
5
6
>>> boucle(2) # ne fait rien puisque 2-1=1 est inférieur à 2
>>>
```

On teste notre programme à partir des éléments de la docstring :

```
>>> est_premier(15)
False
>>> est_premier(17)
True
>>> est_premier(1)
True
```

On a un problème avec 1 qui est un cas particulier à traiter :

On modifie donc le programme :

```
def est_premier(nombre):
    '''
    nombre : int
    déterminer si un entier naturel est premier
    return : True ou False (bool)
    >>>est_premier(15)
    False
    >>>est_premier(17)
    True
    >>>est_premier(1)
    False
    '''
    if nombre==1: # si le nombre est égal à 1
      return False # on renvoie False
    # on utilise une boucle for pour parcourir tous les nombres
    # de 2 à nombre-1
    for diviseur in range(2,nombre) :
        # on calcule le reste de la division euclidienne
        # du nombre par le potentiel diviseur
        if nombre%diviseur==0: # si le reste est égal à 0
```

```
            return False # le nombre n'est pas premier, on renvoit False
    # si on n'a pas trouvé de diviseur en parcourant la boucle
    return True # le nombre est premier, on renvoit False
```

Deuxième approche : on essaie d'optimiser le programme pour diminuer le nombre d'opérations.

Pour déterminer si un programme est efficace, on peut s'intéresser au nombre d'opérations qu'il effectue au maximum, on parle **d'ordre de complexité** du programme.

Notre programmme pour déterminer si un nombre premier n est premier va effectuer un certain nombre d'opérations : n-2

Exemple pour déterminer que n=5 est premier, il va tester la division par 2, puis par 3 et enfin par 4 soit 5-2 = 3 opérations.Pour n très grand, on dira qu'il va effectué environ n opérations. On parle de **complexité linéaire**.

Si l'on réfléchit un peu, on peut améliorer notre programmme en ne testant que les nombres compris entre 2 et la racine carrée de n.

On obtient le programme suivant :

```
def est_premier(nombre):
    '''
    nombre : int
    déterminer si un entier naturel est premier
    return : True ou False (bool)
    >>>est_premier(15)
    False
    >>>est_premier(17)
    True
    >>>est_premier(1)
    False
    '''
    if nombre==1: # si le nombre est égal à 1
        return False
    racine=int(nombre**0.5) # on calcule la partie entière de la racine carrée du
nombre
    # on utilise une boucle for pour parcourir tous les nombres
    # de 2 àracine carrée+1
    for diviseur in range(2,racine+1) :
        # on calcule le reste de la division euclidienne
        # du nombre par le potentiel diviseur
        if nombre%diviseur==0: # si le reste est égal à 0
            return False # le nombre n'est pas premier, on renvoit False
    # si on n'a pas trouvé de diviseur en parcourant la boucle
    return True # le nombre est premier, on renvoit False
```

Pour déterminer que :

```
>>> est_premier(2000003)
True
```

le programme n'effectuera plus environ 2 000 000 d'opérations, mais seulement environ $\sqrt{2000000}$ calculs soit 1 414 opérations ce qui est un gain très appréciable.