Démarrage

débranché

papier & crayon

1. Variables, fonctions et instruction conditionnelle

MÉMO

Quand on programme en Python, on utilise des variables, des fonctions et des instructions. Les variables peuvent être de types différents : numériques entières, numériques flottantes (rationnelles ou réelles), listes, chaînes de caractères ou booléennes.
Ne pas oublier l'indentation, c'est-à-dire le décalage du texte vers la droite, qui indique le début et la fin des instructions conditionnelles, des boucles et des instructions de la fonction.

	Langage naturel	Python
Variable		
De type numérique	*rayon* ← 2	`rayon=2`
De type chaîne de caractères	jour ← "*lundi*"	`jour="lundi"`
De type booléen	*resultat* ← *vrai*	`resultat=True`
Instruction conditionnelle		

	Si condition1 Faire
	.....
	Sinon Si condition2 Faire
	.....
	Sinon Faire
	.....
	Fin Si
		`if condition1:`
		`  ...`
		`elif condition2:`
		`  ...`
		`else:`
		`  ...`

Fonction informatique		

Chaque fonction possède un **nom**, renvoie un **résultat** et peut avoir aucun, un ou plusieurs **paramètres**.		`def nom(paramètre1,paramètre2,...):`
		`  Instructions`
		`  ...`
		`  return resultat`

1 Une fondation attribue des bourses à des étudiants selon leur revenu imposable. Si ce revenu est supérieur ou égal à 30 000 € par an, l'étudiant n'a pas de bourse. Si le revenu est supérieur ou égal à 15 000 € et strictement inférieur à 30 000 € par an, le montant annuel de la bourse est égal à 1 500 € auquel on retranche 5 % du revenu. Si le revenu imposable est strictement inférieur à 15 000 €, le montant est égal à 1 200 € auquel on retranche 3 % du revenu.
Compléter la fonction bourse ci-dessous, qui détermine le montant annuel de la bourse en fonction du revenu de l'étudiant.

```
def bourse(revenu):
    if ..... :
        montant=.....
    elif ..... :
        .....
    else :
        .....
    return .....
```

2 On considère la fonction f définie par le programme ci-contre.

```
def f(x):
    if x>=2:
        y=3*x-4
    elif -1<x<2:
        y=-x+4
    else:
        y=2*x+7
    return y
```

1. Compléter l'expression de $f(x)$ ci-dessous.

$$f(x)=\begin{cases} \ldots\ldots \text{ si } x \in \ldots\ldots \\ \ldots\ldots \text{ si } x \in \ldots\ldots \\ \ldots\ldots \text{ si } x \in \ldots\ldots \end{cases}$$

2. Modifier la fonction ci-dessus pour qu'elle retourne l'image d'un réel x par la fonction f définie pour tout réel x par $f(x)=\begin{cases} x^2-1 \text{ si } x \in [-1;+\infty[\\ 3x+3 \text{ si } x \in]-\infty;-1[\end{cases}$.

CORRIGÉ

3

1. On considère la fonction ci-dessous.

```
def test(n):
    resultat=False
    if n%3==0:
        resultat=True
    return resultat
```

Recopier le script et écrire le résultat ci-dessous.

a. `>>> test(4697)`

b. `>>> test(4697643)`

c. Que teste cette fonction ?

2. On a modifié la fonction précédente comme ci-dessous.

```
def test2(n):
    return n%3==0
```

Recopier le script et écrire le résultat ci-dessous.

a. `>>> test2(4697)`

b. `>>> test2(4697643)`

3. Expliquer ce que renvoie la fonction test2.

4

On considère le programme ci-dessous écrit avec deux fonctions.

```
def f(x):
    return 2*x-4

def signe(f,x):
    if f(x)>=0:
        resultat="positif"
    else:
        resultat="négatif"
    return resultat
```

1. Combien la fonction signe possède-t-elle de paramètres ? Lesquels ?

2. Quel est le résultat renvoyé par l'instruction `>>> signe(f,6)` ?

3. Que réalise la fonction signe ?

4. On écrit dans la console l'expression suivante.

```
>>> signe(lambda x : x*x+2*x-5,7)
```

Que permet l'instruction lambda ?

5. Écrire le script d'une fonction que l'on appellera racine qui renvoie comme résultat la valeur $\sqrt{f(x)}$ pour une fonction f donnée calculée en un réel x donné lorsque le calcul est possible et renvoie comme résultat le mot 'non défini' sinon. On n'oubliera pas d'importer la fonction sqrt de la bibliothèque math.

6. On donne f définie pour tout réel x par :

$$f(x) = -3x + 4.$$

Utiliser la fonction racine pour déterminer les valeurs de $\sqrt{f(x)}$ avec les valeurs de x données.

a. $x = -5$

b. $x = 10$

SPECIMEN

2. Boucles bornées et non bornées

MÉMO

Pour écrire certains programmes, il est parfois utile de répéter une ou plusieurs instructions **un nombre défini de fois**. Lorsque le nombre de répétitions est connu à l'avance, on utilise une **boucle bornée *for***.
Pour d'autres programmes, il est parfois nécessaire de répéter une ou plusieurs instructions **un nombre inconnu de fois**. Lorsque le nombre de répétitions n'est pas connu à l'avance, on utilise une **boucle non bornée** qui est parcourue jusqu'à ce qu'une certaine condition ne soit plus vérifiée. Tant que cette condition est vérifiée, la boucle continue.

	Langage naturel	Python
Boucle bornée		

Boucle avec des valeurs numériques	Pour *variable* allant de ... à ... Faire
	  Instructions
	Fin Pour
		`for variable in range(n):`
		`  instructions`
		`for variable in range(m,n):`
		`  instructions`
		`for variable in range(m,n,p):`
		`  instructions`

Boucle dans une chaîne de caractères	Pour *caractere* dans *chaine* Faire Instructions Fin Pour	`for caractere in chaine:` `  instructions`
Boucle non bornée		
	Tant que condition Faire ... Fin Tant que	`while condition :` `  instructions`

1 On veut calculer une valeur approchée de la somme $1+\frac{1}{3}+\left(\frac{1}{3}\right)^2+\ldots+\left(\frac{1}{3}\right)^{20}$. Compléter la fonction pour qu'elle renvoie cette somme.

```
def somme():
    S=1
    for i in range(  ,  ):
        
    return S
```

2 Compléter la fonction suivante qui compte le nombre d'espaces dans une phrase écrite par l'utilisateur.

```
def compte_espace(phrase):
    nbre=0
    for          in         :
        if caractere==" ":
            nbre=
    return nbre
```

3 On considère la fonction ci-dessous.

```
from random import randint
def nombre_Pile():
    nbre=0
    Pile=0
    while Pile!=1:
        Pile=randint(0,1)
        nbre=nbre+1
    return nbre
```

Expliquer ce que fait la boucle non bornée while.

SPECIMEN

4 On veut calculer, pour tout entier naturel $n > 0$, la somme S égale à :

$$S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots + (-1)^{n+1} \times \frac{1}{n}.$$

Compléter la fonction somme ci-dessous.

```
1  def somme(n):
2      S=....
3      for k in range(...,.....):
4          .....
5      return ....
```

5 Dans une culture bactérienne constituée de N bactéries, on suppose qu'à chaque seconde, 2 % des bactéries meurent.

© Pornsuda Thaisa/Shutterstock

1. Écrire une fonction bacteries qui renvoie le nombre de bactéries encore vivantes après n secondes, n entier naturel.

2. Une culture contient 1 million de bactéries. Combien de bactéries sont encore vivantes après 10 secondes ? Arrondir le résultat à l'unité.

3. En écrivant une fonction appelée moitie, déterminer le nombre de secondes qu'il faut au minimum pour que le nombre de bactéries soit strictement inférieur à la moitié de la population initiale.

6 La conjecture de Syracuse est non encore démontrée à ce jour. Elle s'énonce comme suit :
Soit n un entier naturel. Si n est pair, on le divise par 2, sinon on le multiplie par 3 et on ajoute 1. On recommence ce processus avec le résultat obtenu. La conjecture de Syracuse affirme que l'on obtient toujours 1 au bout d'un nombre fini d'itérations.

1. Compléter la fonction Syracuse qui répond à la conjecture et la tester pour plusieurs valeurs de n.

```
1  def Syracuse(n):
2      resultat=n
3      while resultat!=....:
4          if resultat%2==....:
5              resultat=resultat//2
6          else:
7              .....
8      return resultat
```

2. Modifier la fonction Syracuse en une fonction Syracuse2 afin de compter le nombre d'itérations nécessaires jusqu'à l'obtention de 1. Ce nombre d'itérations s'appelle le temps de vol de n.

3. Soit N un entier donné.
Écrire une fonction Maximum qui renvoie l'entier n entre 1 et N pour lequel le temps de vol est le plus grand et le temps de vol correspondant.

SPECIMEN