ALGORITHMIQUE
ET PROGRAMMATION
EN PYTHON

1. Variables et opérations

▶ Cours

Un programme informatique est une suite d'instructions qu'on donne à l'ordinateur pour qu'il réalise une tâche à notre place.

Un programme utilise très souvent des variables pour réaliser cette tâche.

• Une variable est comme une boîte qui permet de stocker des valeurs (nombre, suite de caractères, réponse tapée au clavier, etc.)

• Pour stocker une valeur dans une variable, on écrit une instruction d'affectation : on affecte une valeur à la variable.

• En Python, l'affectation des variables se fait avec le signe « = » ; on utilise le modèle d'instruction suivant : `nom_de_la_variable = valeur` (on peut mettre des espaces avant et/ou après le signe égal).

Par exemple, l'instruction `longueur=5.7` affecte à la variable « *longueur* » la valeur décimale 5,7.

Après cette instruction, si Python rencontre la variable « *longueur* », il la remplacera par 5,7.

• **Remarque** : le nom d'une variable ne peut pas contenir d'espace. Remplacez cette espace par un tiret dela touche 8 (un underscore) ou utilisez le style « *camelCase* » : par exemple, `nomDeLaVariable = 5.7`.

• Le tableau ci-dessous montre les principales opérations qu'on peut faire avec des variables numériques.

Les signes `>>>` sont l'invite de commande de Python : après ces signes, il attend que vous tapiez une instruction, suivie de la touche Entrée.

Python effectue ensuite la commande que vous avez tapé et affiche éventuellement le résultat.

Affectation des variables	Addition	Soustraction	Produit	Quotient décimal
`>>> a=22` `>>> b=6`	`>>> a+b` `28`	`>>> a-b` `16`	`>>> a*b` `132`	`>>> a/b` `3.666666666666665`

Puissance	Racine carrée	Reste de la division entière de a par b	Quotient de la division entière de a par b

`>>> a**3`
`10648`
	`>>> from math import sqrt`
	`>>> sqrt(a)`
	`4.60941575982343`
		`>>> a%b`
		`4`
			`>>> a//b`
			`3`

▶ Exercices

1 Dans chaque cas, écrire le résultat de l'opération en face de la dernière instruction.

1.
```
>>> longueur=5
>>> largeur=10
>>> longueur*largeur
```
.....

2.
```
>>> from math import pi
>>> rayon = 3
>>> pi*rayon**2
```
.....

2 Après ces instructions, quelle est la valeur de la variable `energie` ?

```
>>> masse = 50
>>> vitesse = 512
>>> energie = 0.5*masse*vitesse**2
```

3 Le loyer mensuel d'un appartement est de 500 € au cours de l'année 2018. Il augmente de 5 % au 1er janvier 2019.

Compléter les instructions suivantes :

```
>>> loyer = 500
>>> taux = 0.05
>>> augmentation =
>>> nouveau_loyer =
```

Qu'affichera Python si on tape `nouveau_loyer` à la suite du prompt `>>>` ?

4 À la suite de ces instructions, quelle est la valeur affichée dans la console Python ?

```
>>> x=4
>>> x=x+6
>>> x=x**2
>>> x
```

5 En 2017, environ 94 % de la population française possédait un téléphone portable (source : CRÉDOC). La population française est estimée à 66 990 826 habitants en 2017 (source : INSEE).

Dans la console Python, créer une variable `taux`, une variable `habitants`, puis utiliser ces variables pour calculer combien de personnes avaient un téléphone portable en France en 2017.

Recopier les instructions ci-dessous :

```
>>> .....
>>> .....
>>> .....
```

2. L'affichage dans la console Python : *print()*

▶ Cours

La fonction `print` est une des plus importantes lorsqu'on crée des programmes pour la console Python (des programmes non-graphiques).

Cette fonction affiche la valeur de ce qui lui est donné en paramètre entre ses parenthèses. Si on veut que `print` affiche plusieurs valeurs, il faut les mettre dans les parenthèses du *print*, séparées par des virgules.

Exemples

Instructions	Signification	Affichage
`nombre = 7` `print(nombre)`	Affiche la valeur de la variable `nombre`	`7`
`nombre = 7` `print("nombre")`	Affiche le texte entre guillemets. Les guillemets peuvent être simples (touche 4 du clavier) ou doubles (touche 3).	`nombre`
`nombre = 7` `print("la valeur de nombre est", nombre)`	Affiche le texte entre guillemets, puis la valeur de la variable `nombre`.	`la valeur de nombre est 7`

`nombre = 7`
`mot = "est plus petit que"`
`valeur = 5`
`print(nombre,mot,valeur)`
	Affiche à la suite (séparées par des espaces) les valeurs des variables `nombre`, `mot` et `valeur`.	`7 est plus petit que 5`

▶ Exercices

1 Quel est l'affichage dans la console Python pour chacun des scripts suivants ?

1.
```
age=16
print(age)
```

2.
```
age=16
print("age")
```

3.
```
age=16
print('Mon âge est', age, 'ans.')
```

4.
```
age=16
print("Je serai centenaire dans", 100-age, "ans.")
```

2 Charlie Chaplin a réalisé le film « *Le Dictateur* » en 1940.

Compléter le script python ci-dessous pour qu'il affiche :

```
Le réalisateur est Charlie Chaplin
Le titre est Le Dictateur
L'année de sortie est 1940
```

.

```
realisateur = "....................
film = "....................
annee = ....................
....................
....................
....................
```

3. L'entrée de valeurs par l'utilisateur : *input()*

▶ Cours

Dans la plupart des programmes, il est nécessaire de demander des renseignements (des valeurs) à l'utilisateur du programme. Cela permet une interaction entre le programme et l'utilisateur.

Grâce à cela, l'utilisateur du programme peut modifier le comportement du programme sans modifier le programme lui-même : les entrées de valeurs permettent à un non-informaticien de se servir quand même du programme.

• La fonction de base pour les entrées est `input()`. Entre les parenthèses de cette fonction, on peut mettre un message qui sera affiché à l'écran, pour indiquer à l'utilisateur qu'il doit entrer une valeur (ce message n'est pas obligatoire).

• La fonction `input()` renvoie comme valeur les caractères tapés par l'utilisateur au clavier (et suivis par la touche Entrée).

Cependant, souvent, c'est une valeur numérique qu'on veut obtenir de l'utilisateur : par exemple, on veut récupérer le nombre 48, pas le caractère 4 suivi du caractère 8.

Pour cela, on utilise autour de `input()` des fonctions de conversion de type : elles permettent de passer du type de données « *caractères* » à un type « *nombre entier* » ou « *nombre décimal* » (Python fait la différence entre ces deux types).

• Pour obtenir un nombre entier, qu'on met dans une variable, on utilise `n = int(input("message"))`.

La valeur numérique de ce qui a été entré au clavier est mise dans la variable `n`. On peut évidemment utiliser un autre nom de variable, et faire autant de lecture de valeurs qu'on veut.

Par exemple, l'instruction `duree=int(input("Entrez le nombre de jours de location : "))` affiche à l'écran le message « *Entrez le nombre de jours de location :* » puis attend que l'utilisateur tape au clavier sa réponse (suivie de Entrée).

Ce qui a été tapé est converti en nombre, et mis dans la variable `duree`, qui est alors disponible dans la suite du programme (pour faire un calcul de coût de location par exemple).

Si l'utilisateur tape quelque chose qui ne peut pas être transformé en nombre (si il met des lettres dans sa réponse par exemple), Python arrêtera le programme avec un message d'erreur.

• Pour obtenir un nombre décimal, on utilise la fonction `float()` (le nombre décimal doit comporter un point, pas une virgule).

Par exemple, `taux=float(input("Entrez le taux de diminution, entre 0 et 1 : "))`.

Il est possible de taper un nombre entier pour une question qui demande un nombre décimal, cela ne provoquera pas d'erreur.

▶ Exercices

1 Que fait le script suivant ?

```
longueur=float(input("Entrez la longueur (en cm) :"))
largeur=float(input("Entrez la largeur (en cm) :"))
print("L'aire est", longueur*largeur, "cm²")
```

2 Écrivez un script qui permette de calculer la longueur de l'hypoténuse d'un triangle rectangle. Le programme doit demander la longueur d'un des côtés de l'angle, puis la longueur de l'angle droit, et enfin afficher avec un message la longueur de l'hypoténuse.

3 Un opticien décide de faire des réductions sur les montures de lunettes d'un pourcentage égal à l'âge du client. Par exemple, un client de 22 ans aura 22 % de réduction sur une monture de lunettes.

Complétez le script suivant pour qu'il demande l'âge du client (c'est un nombre entier), puis le prix initial de la monture (en euros, ce sera donc un nombre décimal car il peut y avoir des centimes), et qu'il affiche le prix réduit.

```
age = int(.....
prix_initial = .....
prix_reduit = .....
print("Le prix après réduction est", .....
```

4 On prouve en sciences physique que la distance d'arrêt d'un véhicule lors d'un freinage d'urgence peut être calculé par la formule $d = 0{,}006v^2 + \frac{5}{18}v$, où d est la distance exprimée en mètres et v la vitesse en km/h.

Écrire un script qui demande la vitesse du véhicule (nombre entier exprimé en km/h), puis qui calcule sa distance d'arrêt, et qui l'affiche avec un message.

Vérifiez avec votre script qu'un véhicule qui roule à 50 km/h s'arrêtera en environ 28,9 m.

4. L'instruction conditionnelle : *if ... : else : ...*

▶ Cours

• Un programme doit parfois réagir de manière différente suivante les valeurs entrées au clavier par l'utilisateur, ou suivant le résultat d'un calcul ou d'un test.

• Tous les langages de programmation permettent d'aiguiller le déroulement d'un programme suivant le résultat de tests (aussi appelés conditions).

Un test est une expression booléenne : c'est une expression qui a pour résultat « *vrai* » (True en python) ou « *faux* » (False). Ce sont les deux seules valeurs possibles pour un test.

```
if condition :
  intruction1
  intruction2
  intruction3
else :
  intruction4
  intruction5
```

• En Python, la syntaxe d'une instruction conditionnelle est montrée dans le script à gauche. Remarquez les points suivants :

1°) la ligne de test commence par *if*, et la condition est suivie d'un deux points (indispensable, sinon Python arrêtera le script avec une erreur de syntaxe).

2°) les instructions 1, 2 et 3 sont celles qui seront exécutées si la condition est vraie (True). Il y en a 3 dans ce script, mais on peut n'en mettre qu'une ou en mettre 15 si necéssaire.

3°) Ces instructions doivent être indentées (décalées vers la droite) du même nombre d'espace pour indiquer qu'elles font partie du même *if*. Ce sont ces indentations qui permettent à Python de choisir les instructions à exécuter lorsque la condition est vérifiée.

Sur papier, quand on écrit un script, on utilise parfois le symbole ␣ qui représente un espace, afin de bien faire voir l'indentation des lignes.

4°) La partie « *else instruction4 instruction5* » est facultative. Ces instructions seront exécutées seulement si la condition qui suit le *if* est fausse (False).

5°) Le mot *else* doit être suivi d'un deux-points ; il doit être aligné avec le *if* auquel il correspond.

Après le *else*, les instructions qui font partie de ce cas doivent être décalées, et alignées les unes avec les autres.

6°) Lorsqu'on veut enchaîner plusieurs tests, on peut utiliser l'instruction *elif*, qui est la contraction de *else if*. Cette instruction doit être alignée avec le *if* correspondant, et elle doit être suivie d'une condition, puis d'un deux-points et après d'instructions indentées, comme pour un *if*. Les instructions seront exécutées si la condition qui suit le *if* est fausse, mais que la condition qui suit le *elif* est vraie.

▶ Exercices

1 Complétez logiquement la ligne qui est dans la partie *else* de ce test :

```
if x<= 4 :
␣␣print("x est inférieur ou égal à 4")
else :
␣␣print("..........................................
```

2 Julien souhaite s'inscrire dans un club d'équitation. Le club lui propose les deux tarifs suivants :

• Tarif A : une cotisation annuelle de 80 € et chaque séance coûte ensuite 10,50 €.

• Tarif B : pas de cotisation annuelle, mais chaque séance coûte 17 €.

Pour prévoir quel tarif serait préférable suivant le nombre de séances qu'il choisira de faire, il a commencé à un script qui demande le nombre (entier) de séances, puis calcule le prix par chacun des tarifs. Terminez son script pour qu' affiche enfin quel est le tarif le plus avantageux.

```
n=int(input("Entrez le nombre de séances :")
tarifA = 80+10.5*n
tarifB = 17*n
if ..........................  :
␣␣print("..........................................
else :
␣␣print("..........................................
```

3 Dans une école de rugby, il y a quatre groupes :

• U8, pour les joueurs entre 8 ans inclus et 10 ans exclus ;

• U10, pour les joueurs entre 10 ans inclus et 12 ans exclus ;

• U12, pour les joueurs entre 12 ans inclus et 14 ans exclus ;

• U14, pour les joueurs entre 14 ans inclus et 16 ans exclus.

Complétez ce script, qui demande l'âge d'un joueur (nombre entier), et qui affiche ensuite à quel groupe ce joueur appartiendra.

```
age = int(input("Donner l'âge du joueur :"))
if age < 8 :
␣␣print("Trop jeune")
elif 8 <= age < 10 :
␣␣print("Groupe U8")
elif ..............................␣ :
␣␣print("..........................................
elif ..........................................
␣␣..........................................
elif ..........................................
␣␣..........................................
else :
␣␣print("Trop âgé")
```

5. Les conditions

▶ Cours

• Lorsqu'on veut tester l'égalité de deux quantités, il faut utiliser un double signe égal : `==`

Le signe égal seul ne peut être utilisé que pour affecter une valeur à une variable.
Mettre un seul signe égal dans une condition (après un `if`) donnera une erreur.

• Pour tester qu'une quantité n'est pas égale à une autre, on utilise un point d'exclamation suivie d'un signe égal : `!=`.

Cela ressemble à un égal barré, qui signifie « *n'est pas égal à* ».

• On peut mettre plusieurs conditions ensembles, en utilisant les mots `and` (et) ou `or` (ou).

Par exemple, pour tester si un nombre x est inférieur à 3 ou supérieur à 5, on mettra la condition `x<3 or x>5`.

• On peut aussi utiliser le mot `not` devant une condition, afin d'en prendre la négation.

Par exemple, pour tester si un nombre x est inférieur à 3 ou supérieur à 5, on peut mettrela condition `not 3<=x<=5` : ce test renverra True si la condtion `3<=x<=5` est False, donc dans le cas où x n'est pas entre 3 et 5.

6. La boucle bornée : *for ... :*

▶ Cours

• Très souvent, un programme doit réaliser une même chose plusieurs fois, qui peut être très grand. Plutôt que de devoir copier-coller de nombreuses fois les instructions à répéter, on utilise une **boucle**.

```
for variable in liste :
␣␣instruction1
␣␣instruction2
␣␣etc.
```

• La syntaxe d'une boucle *for* est montrée à gauche.

Comme pour une instruction conditionnelle *if*, la ligne qui commence par un *for* doit se finir par un deux-points.

Il faut aussi que les instructions que la boucle *for* va répéter plusieurs fois soient indentées (toutes avec le même décalage) par rapport au mot *for*.

• Après le mot *for* vient un nom de variable. Cette variable peut ne pas exister avant le *for*, auquel cas elle sera automatiquement créée.

• Après le nom de variable vient le mot *in*, qui est suivi d'une liste. Comme son nom l'indique, c'est une liste de valeurs, qui sont délimitées par des crochets et séparées par des virgues.

Sur le clavier, on obtient les crochets avec les touches Alt Gr 5 pour le crochet ouvrant, et Alt Gr) pour le crochet fermant.

Un exemple de liste pouvant apparaître après le mot *in* serait `[1,2,3,4,5,6,7,8,9,10]`. Ou bien, par exemple, `[2,3,5,7,11,13,17]`.

Une liste peut aussi contenir des mots, à condition de les mettre entre guillemets simples ou doubles (par exemple, le début de l'alphabet est représenté par la liste `['a', 'b', 'c', 'd', 'e', 'f']`.

Si on met dans une liste un mot non entouré de guillemets, Python considère que c'est un nom de variable, et met dans la liste la valeur de la variable à la place de son nom.

• Quand Python rencontre l'instruction `for variable in liste`, il fait prendre à la variable chacune des valeurs de la liste, et il exécute à chaque fois les intructions qui sont indentées par rapport au mot *for*.

• Python propose une fonction `range` qui génère de manière simple une liste de nombres.

1°) avec l'instruction `range(8)`, Python crée la liste `[0,1,2,3,4,5,6,7]`.

Une liste créée par `range(n)` commence à 0 et va jusqu'à $n-1$ de 1 en 1. Il y a alors n nombres entiers dans cette liste.

2°) avec `range(3,9)`, Python crée la liste `[3,4,5,6,7,8]` : de 3 à 8 par pas de 1 (le 9 n'est pas inclus dans la liste créée).

3°) avec `range(2,14,3)`, Python crée la liste allant de 2 à 14 par pas de 3. Il ne mettra de toute façon pas 14 dans la liste, même si on l'atteint bien en allant de 3 en 3 à partir de 2. On obtient donc la liste `[2,5,8,11]`

4°) avec `range(14,2,-3)`, Python créera la liste `[14,11,8,5]` : de 14 à 2 en allant de -3 en -3, mais 2 sera de toute façon exclu de la liste.

• On peut utiliser n'importe laquelle de ces formes de l'instruction `range` dans une boucle `for`.

Par exemple, `for i in range(4) :` fait que, dans les instructions du bloc qui suit le `for`, i prendra les valeurs 0, 1, 2 et 3.

▶ Exercices

1 Compléter le script ci-dessous pour qu'il affiche la table de multiplication de 13 :

```
for i in range(.....) :
␣␣print("13 x",i,"=", .....)
```

2 Le script ci-dessous calcule et affiche un nombre.

À quelle opération ce nombre correspond-il ?

```
s=0
for i in range(101) :
␣␣s = i + s
print(s)
```

3 Jean a placé 5 000 € sur un compte bancaire rémunéré à 2,5 % par an.

1. Compléter le script suivant pour qu'il calcule et affiche la somme qu'il aura sur son compte après 10 ans d'augmentation.

```
c = .....
for i in range(.....) :
␣␣c = ..........................
print(round(c,2))
```

Remarque : la fonction `round(c,2)` renvoie l'arrondi à 2 chiffres de c.

2. Modifier le script pour qu'il demande le nombre n d'années d'augmentation, puis calcule et affiche la somme sur le compte de Jean après ces n années.

7. La boucle non bornée : *while ... :*

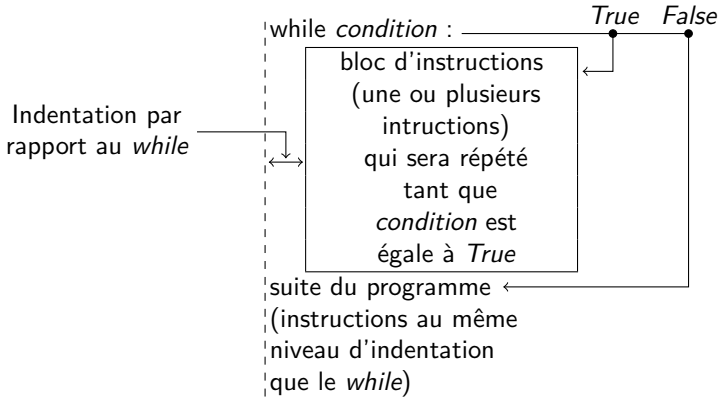
▶ Cours

• La boucle *while* est non bornée dans le sens où on ne sait pas a priori combien de fois elle va être exécutée.

• Après la boucle *while*, on met une condition (un test booléen, qui renvoie *True* ou *False*).

Si le test renvoie *True*, alors les instructions qui sont indentées après le *while* seront exécutées.

Sinon, ces instructions sont passées, et le programme se poursuit après le bloc qui est contenu dans la boucle *while* :



• On peut faire une boucle infinie avec *while True :* car dans ce cas la condition est *True*, dont la valeur est toujours *True*...

Pour sortir d'une telle boucle, on utilise l'instruction *break*.

▶ Exercices

1 Un jardinier souhaite creuser un puits dans son jardin. Le prix du forage dépend de la profondeur atteinte. Le premier mètre coûte 100 €, puis chaque mètre supplémentaire coûte 20 € (120 € pour aller à 2 m, 140 € pour aller à 3 m, etc.)

Pour ce forage, le budget du jardinier est de 700 €. Compléter le script suivant pour qu'il calcule et affiche la profondeur maximale atteinte.

```
profondeur = 1
prix = 700
while prix <= ..... :
    prix = prix + .....
    profondeur = profondeur + .....
print(".....
```

2 On peut générer des nombres aléatories à l'aide du module `random` et de la fonction `randint`. En mettant au début du script l'instruction `import random`, on peut plus loin donner l'instruction `a = random.randint(1,6)` qui affectera à la variable a un nombre aléatoire entre 1 inclus et 6 inclus.

Compléter ce script pour qu'il compte et affiche combien de fois il faut lancer deux dés pour obtenir 12 en faisant la somme des points des dés.

```
import random
somme = 0
nombre = 0
while somme ..... :
    de1 = randint(.....)
    de2 = .....
    somme = .....
    nombre = .....
print(".....
```

3 Réaliser le programme du « plus grand–plus petit » : le script commence par tirer un nombre entier au hasard entre 1 et 1000.

Puis, il demande à l'utilisateur d'entrer un nombre. Le script affiche « *trop grand* » si le nombre de l'utilisateur dépasse celui à deviner, « *trop petit* » sinon.

Ce processus doit se répéter jusqu'à ce que l'utilisateur devine le bon nombre.

Dans ce cas, le script affiche le nombre de propositions que le joueur a faites.

Variante : on peut imposer un nombre maximum de propositions, au-delà duquel le joueur perd le jeu.

8. Les fonctions : *def ...*

▶ Cours

• Dans un programme, il est possible d'écrire des *sous-programmes* ou *fonctions* : ce sont des parties de codes qui sont exécutées lorsqu'on fait appel à elles.

• Une fonction porte un nom, et peut avoir besoin pour fonctionner de variables qu'on appelle les *paramètres* de la fonction.

• On définit une fonction avant de pouvoir l'appeler à partir d'un autre ultérieur du programme.

• La syntaxe générale de définition d'une fonction est la suivante :

```
def nom_de_la_fonction(paramètre1, paramètre2, etc.) :
    instruction1
    instruction2
    instruction3
    etc.
```

Si la fonction n'a pas besoin de paramètres (on dit aussi dans ce cas qu'elle ne prend pas de paramètres) alors on ne met rien entre les parenthèses : `def nom_de_la_fonction() :`

• Comme pour les variables, le nom d'une fonction ne peut pas contenir d'espaces ou de tirets de la touche 6, ni commencer par un chiffre.

• Par exemple, en testant le programme suivant :

```
somme_carres(a,b) :
    s = a**2+b**2
    return(s)

hyp2=somme_carres(3,5)
print(hyp2)
```

, on verra qu'il affiche 34 (car $3^2 + 5^2 = 34$).

L'instruction `return` indique la valeur retournée par la fonction : c'est le résultat de la fonction.

L'instruction `return` n'est pas obligatoire, on ne la met que si la fonction est sensée retourner une valeur.

▶ Exercices

1 On définit la fonction `exercice1` de la façon suivante :

```
def exercice1(a) :
    return a**2-a+1
```

1. Combien cette fonction a-t-elle de paramètres ? .....

2. Que sera-t-il affiché si on tape les commandes suivantes dans le mode direct de Python ?

```
>>> exercice1(2)      .....
>>> exercice1(5)      .....
>>> exercice1(2.1)    .....
```

2 Écrire une fonction `vitesse` qui prend deux paramètres `distance` (supposé être en km) et `temps` (en heures) et qui retourne la vitesse moyenne en km/h.

3 On considère la fonction suivante :

```
import random
def exercice3() :
    return random.randint(1,6)
```

Dans ce script, la commande `import random` met à disposition du programmeur des fonctions additionnelles de Python, parmi lesquelles `randint`. Cette fonction permet d'obtenir un nombre entier aléatoire pris entre les deux nombres donnés en paramètres de `randint`.

Parmi les nombres suivants, lesquels ne peuvent pas être retournés par la fonction `exercice3` ?

2 **3,1** **7** **1,412** **6** **1** **0**

4 Écrire une fonction `solde` qui prend en paramètres un prix et un taux de pourcentage, et qui retourne le prix soldé.

5 La population d'un village était de 3000 habitants en 2017. En moyenne, la ville perd 2 % de ses habitants chaque année.

3. Écrire une fonction `population` qui prend en paramètre une durée en années, et qui calcule la popuplation du village après cette duréee.

Vérifiez en exécutant votre fonction que l'instruction `population(2)` renvoie la valeur 2881,2.

4. Écrire un programme qui détermine au bout de quelle durée la population sera inférieure ou égale à 1500 habitants, et affiche cette durée.

6 Une patinoire propose deux formules de tarification :

• Formule A : chaque entrée coûte 5,25 €.

• Formule B : on paye un abonnement à l'année de 12 €, et chaque entrée coûte alors 3,50 €.

5. Écrire deux fonctions `tarifA` et `tarifB` pour calculer le prix à payer en fonction du nombre d'entrées.

6. Utiliser ces fonctions pour déterminer au bout de combien d'entrées la formule B est la plus avantageuse.

9. Les chaînes de caractères

▶ Cours

• En informatique, le stockage des données repose sur la circulation du courant électrique. Des transistors sont utilisés pour cela : ce sont de minuscules interrupteurs électriques, dont on peut changer l'état électriquement aussi. Un transistor est soit dans l'état ouvert, soit dans l'état fermé

• Un bit est la conceptualisation de ce transistor : un bit vaut 0 ou 1, suivant que le transistor qu'il représente est ouvert ou fermé. 0 ou 1 sont les deux seules valeurs qu'un bit peut prendre : l'information est donc codée en binaire.

• La mémoire de l'ordinateur est constituée d'octets. Ce sont des groupes de 8 bits ; la valeur d'un octet va donc de 0 (les 8 bits sont à 0) jusqu'à 255 (cas où les 8 bits sont à 1).

• Les caractères sont eux aussi codés en binaires : il existe des tables de conversion, qui indique quel caractère correspond à quel code. Par exemple, dans la table de codage ASCII, un octet de valeur 65 correspond à la lettre majuscule A, 66 correspond à B, 67 à C, etc.

• Comme 255 caractères n'étaient pas suffisant pour pouvoir coder tous les alphabets et les symboles connus, un comité a établi Unicode comme remplacement des tables de codage multiples qu'il y avait auparavant.

Le principe est le même : des octets codent les caractères, mais la table de codage est beaucoup plus grand qu'avant.

• Une chaîne de caractères est un groupe d'octets qui est interprété comme une succession de caractères.

Par exemple, la suite d'octets 65 66 67 68 69 code la chaîne "ABCDE" ; inversement, le mot "CHAINE" sera stocké dans la mémoire de l'ordinateur sous la forme de la suite d'octets

• Python convertit automatiquement les chaînes de caractères en octets.

Par exemple, avec la commande `s="CHAINE"`, Python crée en mémoire la suite d'octets 67, 72, 65, 73, 78, 69.

Mais l'instruction `print(s)` n'affichera pas la suite d'octets, elle affichera la chaîne de caractères, car Python fait dans ce cas la correspondance entre les octets et les caractères

• Pour déterminer la longueur d'une chaîne de caractères, on utilise la commande `len`. Si `s="CHAINE"`, alors la commande `len(s)` renverra 6 comme valeurs (il y a 6 caractères dans s).

On peut mettre des espaces dans une chaîne de caractères. Chaque espace compte comme un caractère.

• Il est possible de faire des opérations avec les chaînes de caractères :

Concaténation : +	Répétition : *	Égalité : ==	Différence : !=	Comparaison lexicographique	

>>> x="auto"
>>> y="bus"
>>> x+y
'autobus'
	>>> x="bla"
	>>> x*3
	'blablabla'
		>>> x="chien"
		>>> x=="chien"
		True
		>>> x=="chine"
		False
		>>> x=="CHIEN"
		False
			>>> x="chien"
			>>> x !="chine"
			True
			>>> x !="CHIEN"
			True
				>>> x="fiole"
				>>> y="folie"
				>>> x<y
				True
					>>> x="foule"
					>>> y="fou"
					>>> x<y
					False

• On peut extraire un caractère d'une chaîne, en fonction de sa position de la chaîne. Pour cela, on met le nom de la variable, suivi entre crochets du numéro du caractère qu'on veut (les caractères sont numérotés à partir de 0).

Par exemple avec la chaîne s égale à : "

B	o	n	j	o	u	r		à		t	o	u	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13

"

Dans ce cas, `s[0]` correspond au 'B', `s[1]` au premier 'o', `s[2]` au 'n', etc.

• On peut aussi prendre des « tranches » d'une chaîne, c'est-à-dire des parties ou sous-chaînes.

Pour cela, on met entre crochet le rang de départ puis un deux-points suivi du rang final.

Par exemple, avec la chaîne `s="Bonjour à tous"`, la commande `s[0 :3]` correspond à "Bon" et `s[4 :9]` correspond à "our à".

• On peut utiliser des nombres négatifs, cela permet de repérer les caractères depuis la fin de la chaîne : `s[-1]` correspond au 's' final, `s[-2]` au 'u' en avant-dernière position, `s[-3]` au 'o', etc.

• On peut utiliser la boucle `for` pour un chaîne de caractères, pour parcourir les caractères de la chaîne un par un.

Par exemple, avec

```
s="Bonjour à tous"
for c in s :
␣␣print(c)
```

, on verra s'afficher un à un les caractères de s.

▶ Exercices

1 La chaîne de caractères x est définie par l'instruction `x="Cet exercice est très intéressant␣!"`.

Complétez les cases du tableau ci-dessous.

Question	Quelle est la longueur de la chaîne ?				Quel est le dernier caractère ?	
Affichage			'x'			
Instruction correspondante		x[14]		x[13 :17]		x[7]+x[17 :20]

2 La chaîne de caractères abc est définie par la commande `abc='trois lettres'`. Complétez les affichages obtenus en réponse aux instructions ci-dessous :

```
>>> print(abc) .....
>>> print('abc') .....
>>> '1'+'2'+'3' .....
>>> 1+2+3 .....
>>> abc+'d' .....
```

3 On a commencé un programme qui demande une phrase à l'utilisateur, puis qui compte combien il y a d'espaces dans cette phrase.

Complétez ce programme et testez-le :

```
phrase = input("Tapez votre phrase␣:␣")
n=0
for c in ..... :
␣␣if c == ..... :
␣␣␣␣n=.....
print("Il y a", n, "espaces dans votre phrase")
```

4 Inspirez-vous du programme de l'exercice précédent pour créer un script qui compte la fréquence de 'e' (majuscules ou minuscules) dans une phrase.

Attention, il faut compter les 'é' 'è' 'ê' et 'ë' comme des 'e' !

Approfondissement : Calculez la fréquence de chacune des voyelles, voire la fréquence de chacune des lettres présente dans le texte.

5 Les fonctions `upper()`, `lower()` et `capitalize()` s'appliquent à une chaîne de caractères.

1. Complétez les exemples suivants pour voir ce qu'elles font.

```
>>> p="Il FAit bEAu auJOUrd'hui␣!
>>> p.upper() .....
>>> p.lower() .....
>>> p.capitalize() .....
```

2. Complétez ce script pour faire la fiche d'identité d'une personne, en mettant son nom en majuscules et son prénom avec une lettre capitale :

```
nom = input("Quel est votre nom ? ")
prenom = input("Quel est votre prénom ? ")
age = input("Quel est votre âge ? ")
print("NOM :", ..... )
print("Prénom :", ..... )
print("Âge :", ..... )
```