TESTS

1. Exemples d'introduction

(a) Proposer un algorithme en langage naturel permettant de trouver le minimum de trois nombres distincts donnés a , b et c .

(b) Le calendrier que nous utilisons est légèrement décalé par rapport à la période de révolution de la terre autour du soleil. Une année possède 365 jours mais, pour la Terre, une année est un peu plus longue de quelques heures. Pour corriger ce décalage, on ajoute un jour à certaines années, appelées années bissextiles.
Ces années bissextiles sont les années des multiples de 4 mais avec une exception : on n'ajoute pas de jour les années multiples de 100. Cependant, les années multiples de 400 sont quand même bissextiles.
Ainsi, l'année 1976 était bissextile, l'année 1979 ne l'était pas, l'année 2000 était bissextile mais l'année 1900 ne l'était pas.

2. Définition

Les **tests** sont un élément essentiel à tout langage informatique si on veut lui donner un peu de complexité car ils permettent à l'ordinateur de prendre des décisions.

Pour cela, Python utilise l'instruction : if ainsi qu'une comparaison parmi celles-ci dessous :

Syntaxe Python	Signification
==	égal à
!=	différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Exemples

	Commentaires

`>>> x = 2`
`>>> if x == 2:`
`...  print("Le test est vrai !")`
`...`
`Le test est vrai !`
	Rappel : En python, « **=** » n'a pas la même signification qu'en mathématiques.
	Le double égal « **==** » pose la question :
	la valeur de la variable x est-elle égale à 2 ?
	Un égal simple aurait affecté 2 à la variable x
	On aurait eu alors un message d'erreur

`>>> x = "souris"`
`>>> if x == "tigre":`
`...  print("Le test est vrai !")`
`...`
	On peut donc comparer tout type de variables.
	Le test étant faux, il n'est pas affiché

Remarques importantes

- Les blocs d'instructions dans les tests doivent forcément être indentés (décalés). L'indentation indique la portée des instructions à exécuter si le test est vrai.
- L'instruction **if** se termine **toujours** par le caractère « **:** »

3. Tests à plusieurs cas

Parfois, il est pratique de tester si la condition est vraie ou si elle est fausse dans une même instruction **if**.
Plutôt que d'utiliser deux instructions **if,** on peut se servir des instructions **if** et **else** :

Exemple 1	**Commentaires éventuels**

`>>> x = 2`
`>>> if x == 2:`
`...  print("Le test est vrai !")`
`... else:`
`...  print("Le test est faux !")`
`...`
`Le test est vrai !`
`>>> x = 3`
`>>> if x == 2:`
`...  print("Le test est vrai !")`
`... else:`
`...  print("Le test est faux !")`
`...`
`Le test est faux !`	

Exemple 2

On se propose de tirer au sort une base d'ADN puis d'afficher le nom de cette dernière.

Dans le code suivant, nous utilisons l'instruction **`random.choice(liste)`** qui renvoie un élément choisi au hasard dans une liste. L'instruction **`import random`** est une instruction qui nous permet d'utiliser l'instruction **`random`** qui nous permet de choisir un élément d'une liste au hasard.

	Commentaires

`>>> import random`
`>>> base = random.choice(["a", "t", "c", "g"])`
`>>> if base == "a":`
`...  print("choix d'une adénine")`
`... elif base == "t":`
`...  print("choix d'une thymine")`
`... elif base == "c":`
`...  print("choix d'une cytosine")`
`... elif base == "g":`
`...  print("choix d'une guanine")`
`...`
`choix d'une cytosine`
	L'instruction **`elif`** nous permet d'utiliser une série de tests dans la même instruction **`if`** pour tester plusieurs valeurs d'une **même** variable (ici la variable **base**).

4. Importance de l'indentation

Tester plusieurs fois les deux codes suivants :

```
import random
base = random.choice(["a", "t"])
if base == "a":
    print("choix d'une adénine")
    print("car le choix est",base)
else :
    print("choix d'une thymine")
    print("car le choix est",base)
```

```
import random
base = random.choice(["a", "t"])
if base == "a":
    print("choix d'une adénine")
    print("car le choix est",base)
else :
    print("choix d'une thymine")
print("car le choix est",base)
```

Commentaires

5. Tests multiples

Les tests multiples permettent de tester plusieurs conditions en même temps en utilisant des opérateurs qu'on appelle booléens.

Les deux opérateurs les plus couramment utilisés sont le **OU** et le **ET**.

Compléter les deux tableaux suivants :

Fonctionnement de l'opérateur OU

Condition 1	Opérateur	Condition 2	Résultat
Vrai	OU	Vrai	
Vrai	OU	Faux	
Faux	OU	Vrai	
Faux	OU	Faux	

Fonctionnement de l'opérateur ET

Condition 1	Opérateur	Condition 2	Résultat
Vrai	ET	Vrai	
Vrai	ET	Faux	
Faux	ET	Vrai	
Faux	ET	Faux	

En Python, on utilise le mot réservé **and** pour l'opérateur **ET** et le mot réservé **or** pour l'opérateur **OU**. Respectez bien la casse des opérateurs and et or qui, en Python, s'écrivent en minuscule.

Exemple

```
>>> x = 2
>>> y = 2
>>> if x == 2 and y == 2:
...     print("le test est vrai")
...
le test est vrai
```

6. Applications

(a) Minimum d'une liste, année bissextile

Coder et tester les algorithmes a) et b)

(b) Notes et mention d'un étudiant (facile)

Un étudiant a obtenu 5 notes comprises entre 0 et 20.

- Créez un script qui affiche la note maximum (utilisez la fonction max()), la note minimum (utilisez la fonction min()) et qui calcule la moyenne.
- Affichez aussi la mention obtenue sachant que la mention est « passable » si la moyenne est entre 10 inclus et 12 exclus, « assez bien » entre 12 inclus et 14 exclus et « bien » au-delà de 14.

(c) Séquence complémentaire d'un brin d'ADN

La liste ci-dessous représente la séquence d'un brin d'ADN : ["A","C","G","T"]
Créez un script qui transforme cette séquence en sa séquence complémentaire.

Rappel

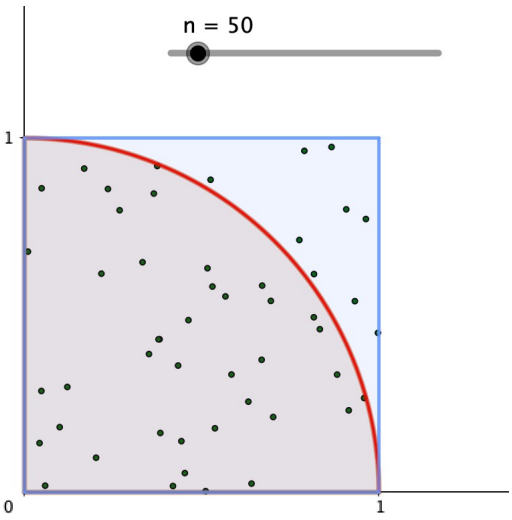
la séquence complémentaire s'obtient en remplaçant "A" par "T", "T" par "A", "C" par "G" et "G" par "C".

(d) Vers l'utilisation des boucles bornées : la méthode de Monte-Carlo (plus difficile)

On se place dans un repère orthonormé $(O;\vec{i},\vec{j})$.

L'une des méthodes pour obtenir une approximation de π consiste à choisir au hasard un grand nombre de points ayant une abscisse et une ordonnée dans l'intervalle $[0;1]$.

La proportion de ceux qui sont à l'intérieur du quart de cercle de centre O et de rayon 1 est une approximation de $\frac{\pi}{4}$.



répartition de 50 points pris au hasard dans le carré de côté 1

1. Écrire un script permettant d'approximer π comportant les variables :
   - **compte** : comptabilise le nombre de points dans le quart de cercle de centre O et de rayon 1.
   - **proportion** : correspond à la proportion de points dans le quart de cercle.
   - **x** et **y** : représentant les coordonnées d'un point choisi au hasard.
   - **n** : le nombre de points à considérer (au choix de l'utilisateur)

   On utilisera l'instruction **random()** qui renvoie un flottant compris entre 0 et 1.

   (pour pouvoir utiliser **random()**, vous écrirez en début de script l'instruction suivante : **import random**).

2. Tester le programme à plusieurs reprises. L'approximation obtenue est-elle fiable ?