



Accueil > Cours > Apprenez à programmer en Python > Quiz : Comprenez les bases du langage Python

Apprenez à programmer en Python

40 heures  Difficile



Mis à jour le 26/06/2019



Comprenez les bases du langage Python

Bravo ! Vous avez réussi cet exercice !

Compétences évaluées

-  Traiter les exceptions et les erreurs
-  Écrire des commandes dans l'interpréteur Python

Question 1

Après ces instructions, de quel type est la variable `c` ?

```
a = 8  
b = 3  
c = a / b
```

- ☐ `int` (entier)
- ☒ `float` (flottant)
- ☐ `str` (chaîne de caractères)

Les variables `a` et `b` sont toutes les deux entières, mais le résultat de la division de `8 / 3` retourne un nombre flottant (quelque chose comme `2,6666...`). En conséquences, c'est le type `float` (flottant) qui est retourné par Python.

Question 2

Quelle est la variable de type `str` (chaîne de caractères) parmi les choix suivants ?

- ☐ 3
- ☐ 3.1
- ✓ ☒ "3"

Une chaîne de caractères (type `str`) est identifié par des apostrophes ou guillemets l'entourant. Les deux premiers choix sont des nombres (type `int` et type `float`).

Question 3

Quelle est la différence entre entrer une variable dans l'interpréteur interactif et utiliser la fonction `print` ?

- ☐ Aucune
- ☐ Dans l'interpréteur interactif, toutes les variables apparaissent entourées de guillemets.
- ✓ ☒ La fonction `print` est dédiée à l'affichage, l'interpréteur au débbugage.

La fonction `print` est spécialement dédiée à l'affichage. Si vous exécutez votre programme Python depuis un fichier, entrer directement la variable n'affichera rien. Cependant, l'interpréteur interactif permet d'entrer une variable directement pour afficher son contenu, bien que le résultat obtenu reste différent de la fonction `print`.

Question 4

De quoi doit être composée une condition au minimum ?

- ✓ ☒ D'un bloc `if`
- ☐ D'un bloc `if` et `elif`
- ☐ D'un bloc `if` et `else`

Une condition peut n'être constituée que d'un unique bloc `if`, sans définir de bloc qui s'exécutera dans le cas où le prédicat n'est pas vérifié. Les blocs `else` et `elif` sont donc optionnels.

Question 5

Considérant les instructions ci-dessous, si `variable` vaut `2.8`, quel va être le résultat obtenu ?

```
if variable >= 3:
    print("1")
elif variable < -1:
    print("2")
else:
    print("3")
```

- ☐ Afficher 1.
- ☐ Afficher 2.
- ☒ Afficher 3.
- ☐ N'afficher rien.

Si `variable` vaut `2.8`, le premier prédicat est incorrect (la variable est inférieure à 3). Le second prédicat est également faux (la variable n'est pas inférieure à -1). C'est donc le bloc `else` qui s'exécute puisque tous les autres ne se sont pas exécutés.

Question 6

Considérant le prédicat combiné ci-dessous, dans quel cas sera-t-il `True` (vrai) ?

`predicat_a and predicat_b`

- ☐ `predicat_a` est vrai, peu importe `predicat_b`.
- ☐ `predicat_b` est vrai, peu importe `predicat_a`.
- ☐ L'un d'eux est vrai, peu importe l'autre.
- ☒ `predicat_a` et `predicat_b` sont tous deux vrais.

Nos deux prédicats (`predicat_a` et `predicat_b`) sont combinés par le mot-clé `and` (et). Ils doivent être tous deux vrais pour que le prédicat combiné soit vrai également.

Question 7

Comment Python identifie-t-il les instructions formant un bloc (par exemple à l'intérieur d'une condition) ?

- ✓ ☒ Grâce à l'indentation
- ☐ Grâce aux accolades (`{ }`) entourant le bloc
- ☐ Grâce aux deux points en début de bloc

Contrairement à de nombreux langages, l'indentation (le décalage de certaines instructions grâce à des espaces ou tabulations) suffit à Python pour identifier un bloc d'instructions. Il a donc besoin d'une indentation cohérente, contrairement à ces mêmes langages qui n'utilisent l'indentation que pour le confort du programmeur.

Question 8

Quel est l'avantage de la boucle `while` sur la boucle `for` ?

- ☐ Aucun.
- ☐ Elle est préférable pour parcourir des séquences.
- ☐ Elle fait la même chose en moins de lignes de code.
- ☐ Elle crée rarement de boucle infinie.
- ✓ ☒ Elle est plus utile pour vérifier une condition.

La boucle `while` est moins utilisée en Python que la boucle `for`, cette dernière étant plus utile pour parcourir des séquences, ce qui rend le second choix faux. Notez que la boucle `while` peut également parcourir des séquences, bien que le code résultant soit moins intuitif et souvent plus long, ce qui rend le choix suivant faux également. Enfin, il est à noter que la boucle `while` provoque potentiellement plus de boucles infinies que la boucle `for`. Elle est en effet utilisée pour vérifier une condition (c'est donc le dernier choix qui est vrai), mais la condition utilisée peut rester vraie indéfiniment si le développeur n'est pas prudent.

Question 9

Quel est l'avantage de la boucle `for` sur la boucle `while` ?

- ✓ ☒ Elle est préférable pour parcourir des séquences.
- ☐ Elle est plus utile pour vérifier une condition.

- ☐ C'est l'unique façon de parcourir des séquences.

La boucle `for` est plus utilisée en Python pour parcourir des séquences, alors que la boucle `while` est de préférence utilisée pour vérifier une condition. Notez cependant qu'on peut parcourir des séquences avec `while` dans la grande majorité des cas, même si ce n'est pas le plus intuitif.

Question 10

Quelle est la différence entre le mot-clé `break` et `continue` ?

- ✓ ☒ `break` interrompt la boucle immédiatement alors que `continue` passe au prochain tour de boucle.
- ☐ `continue` interrompt la boucle immédiatement alors que `break` passe au prochain tour de boucle.
- ☐ Les deux mot-clés font exactement la même chose en interrompant la boucle immédiatement.

Le mot-clé `break` interrompt immédiatement la boucle, alors que le mot-clé `continue` passe directement au prochain tour de boucle. Ces deux mot-clés ne font donc pas la même chose.

Question 11

Sachant la définition de fonction ci-dessous, quel est l'appel INVALIDE parmi les choix suivants ?

```
def table(nombre, maximum=10):
```

- ☐ `table(5, 20)`
- ☐ `table(12, maximum=5)`
- ☐ `table(8)`
- ☐ `table(maximum=15, nombre=4)`
- ✓ ☒ `table(7, entier=30)`

Tous les choix proposés sont des appels valides de la fonction, sauf le dernier choix.

- Le premier choix n'utilise que deux paramètres ordonnés (sans les nommer), mais puisqu'il y en a bien deux, la fonction devrait s'exécuter ;
- Le second choix utilise les paramètres ordonnés ou nommés indifféremment ;
- Le troisième choix ne précise qu'un paramètre, mais le second a une valeur par défaut dans la définition de la fonction ;
- Le quatrième choix précise les paramètres dans le désordre, ce qui paraît un peu étrange, mais puisqu'ils sont nommés, tout va bien ;
- Le cinquième choix précise un paramètre nommé entier qui n'est pas présent dans la définition de la fonction. Ce choix est donc invalide.

Question 12

Quelle est l'utilité des fonctions **lambda** par rapport aux fonctions définies par `def` ?

- ☐ Elles s'exécutent plus rapidement.
- ✓ ☒ On peut en créer avec une syntaxe très légère.
- ☐ Elles permettent des fonctionnalités inaccessibles aux fonctions définies par `def`.

L'avantage des fonctions **lambda** est dans leur syntaxe allégée. Elles ne proposent pas de fonctionnalité qu'on ne pourrait utiliser dans une fonction définie par `def` et ne sont pas plus rapides à s'exécuter (en réalité, elles sont même un peu plus lentes).

Question 13

Qu'est-ce qu'un module en Python ?

- ☐ Un fichier contenant du code Python sans extension particulière.
- ✓ ☒ Un fichier contenant du code Python avec l'extension `.py`
- ☐ Un répertoire contenant des fichiers Python

Les modules en Python sont de simples fichiers contenant du code Python. Ils doivent cependant avoir l'extension `.py` pour être reconnus comme des modules par Python.

Question 14

Si vous entrez l'instruction `import azerty`, les fonctions du module `azerty` seront ... ?

- ☐ Directement accessibles en entrant simplement leur nom.
- ✓ ☒ Accessibles en préfixant leur nom par `azerty.`
- ☐ Accessibles en préfixant leur nom par un simple point (`" . "`).

La syntaxe `import azerty` crée un nouvel espace de noms, `azerty`, dans lequel se trouveront les fonctions et données du module. Cela signifie que vous devrez préfixer le nom des fonctions et données du module par `azerty.` dans le code important ce module avec cette syntaxe.

Question 15

En utilisant l'instruction `from ... import *`, que peut-on importer ?

- ☐ Seulement des fonctions d'un module
- ☐ Seulement des modules d'un package
- ✓ ☒ Tout ce que contient un module ou package

La syntaxe `from ... import *` permet d'importer tout ce que contient un module ou package : ce peut être d'autres modules, d'autres packages, des variables, fonctions ou d'autres données qui seront abordées plus loin dans ce cours.

Question 16

Qu'est-ce qui caractérise, au minimum, un package Python ?

- ✓ ☒ Un répertoire simple
- ☐ Un répertoire avec, au minimum, un fichier `__init__.py` dedans
- ✗ ☐ Un répertoire avec un fichier `__init__.py` et au moins un autre module ou package

Dans les versions de Python 3.3 et ultérieures, le fichier `__init__.py` qui contient du code d'initialisation du package est devenu facultatif. Un simple répertoire suffit, tant qu'il se conforme aux noms autorisés par Python. Notez qu'un package peut être complètement vide (sans aucun autre package ou module), même si, en pratique, il est rare d'en trouver un.

Question 17

Quels sont les mot-clés minimums pour capturer une exception ?

- ☐ `try` et `catch`

- ✓ ☒ try et except
- ☐ try et else

En Python, un bloc `try` doit avoir une répartition `except` qui dit à Python quoi faire si le code contenu dans le bloc `try` lève une exception. Le mot-clé `catch` n'existe pas en Python. Quant au mot-clé `else`, il est facultatif.

Notez qu'on peut théoriquement former un bloc `try` qu'avec une répartition `finally`, bien qu'en pratique ce soit assez rare.

Question 18

Quel mot-clé est utilisé pour lever une exception ?

- ☐ throw
- ✓ ☒ raise
- ☐ try

Pour lever une exception en Python, on utilise le mot-clé `raise`. Le mot-clé `throw` n'existe pas en Python, tandis que le mot-clé `try` est utilisé pour essayer un bloc d'instructions (pour récupérer ses exceptions éventuelles), pas pour lever des exceptions.

Question 19

Dans quel cas le bloc `finally` est-il exécuté ?

- ☐ Quand aucune exception ne se produit dans le bloc `try`.
- ☐ Quand une exception se produit dans le bloc `try`.
- ✓ ☒ Dans tous les cas.

Le bloc `finally` s'exécute dans tous les cas, que le bloc `try` ait levé une exception ou non. Faites bien la différence avec le bloc `else`, qui ne s'exécute que dans le cas où il n'y a eu aucune exception de levée.

Question 20

Dans quel cas l'instruction ci-dessous lèvera une exception ?


```
annee = int(entree)
```

- ☐ La variable `annee` n'existe pas.
- ☐ La variable `entree` est une chaîne de caractères.
- ✓ ☒ La variable `entree` ne peut être convertie en nombre.

Cette ligne d'instruction lèvera une exception dans le cas où la variable `entree` ne peut être convertie en nombre. L'existence (ou l'inexistence) de la variable `annee` n'importe pas ici, puisqu'on cherche à écrire dans cette variable, pas lire dedans. Enfin, la variable `entree` peut très bien être une chaîne de caractères, tant qu'elle peut être convertie.

[TP : TOUS AU ZCASINO](#)[CRÉEZ VOTRE PREMIER OBJET : LES
CHAÎNES DE CARACTÈRES](#)

Le professeur

Vincent Le Goff

Découvrez aussi ce cours en...



Livre



PDF

OpenClassrooms

L'entreprise

Alternance

Forum

Blog

Nous rejoindre

Entreprises

Employeurs

En plus

Devenez mentor

Aide et FAQ

Conditions Générales d'Utilisation

Politique de Protection des Données Personnelles

Nous contacter

 Français ▼

