Plan

Python si ... sinon

Conditions Python et instructions If

Python prend en charge les conditions logiques habituelles des mathématiques:

- Égale: `a == b`
- Différent de: `a! = B`
- Moins de: `a <b`
- Inférieur ou égal à: `a <= b`
- Supérieur à: `a> b`
- Supérieur ou égal à: `a> = b`

Ces conditions peuvent être utilisées de plusieurs manières, le plus souvent dans les «instructions if» et les boucles.

Une "instruction if" est écrite à l'aide du mot clé `if` .

Exemple

Si déclaration:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Dans cet exemple, nous utilisons deux variables, `a` et `b` , qui sont utilisées dans le cadre de l'instruction if pour tester si `b` est supérieur
à `a` . Comme `a` vaut `33` et `b` vaut `200` , nous savons que 200 est supérieur à 33, et nous imprimons donc à l'écran que "b est supérieur à a".

Échancrure

Python s'appuie sur l'indentation (espace blanc au début d'une ligne) pour définir la portée dans le code. D'autres langages de programmation utilisent souvent des accolades à cette fin.

Exemple

Si instruction, sans indentation (générera une erreur):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

Elif

Le mot-clé `elif` est une manière pythons de dire "si les conditions précédentes n'étaient pas vraies, alors essayez cette condition".

Exemple

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Dans cet exemple, `a` est égal à `b` , donc la première condition n'est pas vraie, mais la condition `elif` est vraie, donc nous imprimons à l'écran que "a et b sont égaux".

Autre

Le mot clé `else` attrape tout ce qui n'est pas intercepté par les conditions précédentes.

Exemple

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
```

```
  print("a and b are equal")
else:
  print("a is greater than b")
```

Dans cet exemple, `a` est supérieur à `b` , donc la première condition n'est pas vraie, aussi la condition `elif` n'est pas vraie, donc nous passons à la condition `else` et imprimons à l'écran que "a est supérieur à b".

Vous pouvez également avoir un `else`sans le `elif`:

Exemple

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Main courte si

Si vous n'avez qu'une seule instruction à exécuter, vous pouvez la placer sur la même ligne que l'instruction if.

Exemple

Une ligne si instruction:

```
if a > b: print("a is greater than b")
```

Main courte si ... sinon

Si vous n'avez qu'une seule instruction à exécuter, une pour if et une pour else, vous pouvez tout mettre sur la même ligne:

Exemple

Une ligne if else instruction:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

ette technique est connue sous le nom d' **opérateurs ternaires** ou d' **expressions conditionnelles** .

Vous pouvez également avoir plusieurs instructions else sur la même ligne:

Exemple

Instruction if else sur une ligne, avec 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

Et

Le mot-clé `and` est un opérateur logique et est utilisé pour combiner des instructions conditionnelles:

Exemple

Teste si `a`est supérieur à `b`, ET si `c` est supérieur à `a`:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Ou

Le `or`mot-clé est un opérateur logique et est utilisé pour combiner des instructions conditionnelles:

Exemple

Teste si `a`est supérieur à `b`, OU si `a` est supérieur à `c`:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Imbriqué si

Vous pouvez avoir des `if`instructions dans des `if`instructions, c'est ce qu'on appelle *des* `if` instructions *imbriquées* .

Exemple

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

La déclaration de réussite

`if`Les instructions ne peuvent pas être vides, mais si, pour une raison quelconque, vous avez une `if`instruction sans contenu, insérez-la `pass`pour éviter d'avoir une erreur.

Exemple

```
a = 33
b = 200

if b > a:
  pass
```

Boucles While Python

Boucles Python

Python a deux commandes de boucle primitives:

- `while` boucles
- `pour les` boucles

La boucle while

Avec le `tout` en boucle , nous pouvons exécuter un ensemble d'instructions tant qu'une condition est vraie.

Exemple

Imprimez i tant que i est inférieur à 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Remarque: n'oubliez pas d'incrémenter i, sinon la boucle continuera indéfiniment.

Le `tout` en boucle nécessite des variables pertinentes pour être prêt, dans cet exemple , nous devons définir une variable d'indexation, `i` , que nous fixons à 1.

La déclaration de rupture

Avec l' instruction `break` , nous pouvons arrêter la boucle même si la condition while est vraie:

Exemple

Quittez la boucle lorsque i est 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

La déclaration continue

Avec l' instruction `continue` , nous pouvons arrêter l'itération en cours et continuer avec la suivante:

Exemple

Passez à l'itération suivante si i vaut 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

La déclaration else

Avec l' instruction `else` , nous pouvons exécuter un bloc de code une fois lorsque la condition n'est plus vraie:

Exemple

Imprimez un message une fois que la condition est fausse:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Python pour les boucles

Python pour les boucles

Une boucle `for` est utilisée pour itérer sur une séquence (c'est-à-dire une liste, un tuple, un dictionnaire, un ensemble ou une chaîne).

Cela ressemble moins `au` mot-clé `for` dans d'autres langages de programmation, et fonctionne plus comme une méthode d'itération que l'on trouve dans d'autres langages de programmation orientés objet.

Avec la boucle `for` , nous pouvons exécuter un ensemble d'instructions, une fois pour chaque élément d'une liste, d'un tuple, d'un ensemble, etc.

Exemple

Imprimez chaque fruit dans une liste de fruits:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

La boucle `for` ne nécessite pas de variable d'indexation à définir au préalable.

Boucle sur une chaîne

Même les chaînes sont des objets itérables, elles contiennent une séquence de caractères:

Exemple

Parcourez les lettres du mot «banane»:

```
for x in "banana":
  print(x)
```

La déclaration de rupture

Avec l' instruction `break` , nous pouvons arrêter la boucle avant qu'elle n'ait parcouru tous les éléments:

Exemple

Quittez la boucle quand `x`est "banane":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Exemple

Quittez la boucle quand `x`est "banane", mais cette fois la pause vient avant l'impression:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

La déclaration continue

Avec l' instruction `continue` , nous pouvons arrêter l'itération actuelle de la boucle et continuer avec la suivante:

Exemple

N'imprimez pas de banane:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

La fonction range ()

Pour parcourir un ensemble de code un certain nombre de fois, nous pouvons utiliser la fonction `range ()` ,

La fonction `range ()` renvoie une séquence de nombres, commençant par 0 par défaut et incrémentée de 1 (par défaut), et se termine à un nombre spécifié.

Exemple

Utilisation de la fonction range ():

```
for x in range(6):
  print(x)
```

Notez que la `plage (6)` ne correspond pas aux valeurs de 0 à 6, mais aux valeurs de 0 à 5.

La fonction `range () par` défaut à 0 comme valeur de départ, cependant il est possible de spécifier la valeur de départ en ajoutant un paramètre: `range (2, 6)` , ce qui signifie des valeurs de 2 à 6 (mais sans compter 6):

Exemple

Utilisation du paramètre de démarrage:

```
for x in range(2, 6):
  print(x)
```

La fonction `range () par` défaut incrémente la séquence de 1, cependant il est possible de spécifier la valeur d'incrément en ajoutant un troisième paramètre: `range (2, 30, 3 )` :

Exemple

Incrémentez la séquence de 3 (la valeur par défaut est 1):

```
for x in range(2, 30, 3):
  print(x)
```

Sinon dans la boucle For

Le `else`mot-clé dans une `for`boucle spécifie un bloc de code à exécuter lorsque la boucle est terminée:

Exemple

Imprimez tous les nombres de 0 à 5 et imprimez un message lorsque la boucle est terminée:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Remarque: Le `else`bloc ne sera PAS exécuté si la boucle est arrêtée par une `break`instruction.

Exemple

Brisez la boucle quand `x`est 3 et voyez ce qui se passe avec le `else`bloc:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

Boucles imbriquées

Une boucle imbriquée est une boucle à l'intérieur d'une boucle.

La "boucle interne" sera exécutée une fois pour chaque itération de la "boucle externe":

Exemple

Imprimez chaque adjectif pour chaque fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

La déclaration de réussite

`for`les boucles ne peuvent pas être vides, mais si, pour une raison quelconque, vous avez une `for`boucle sans contenu, insérez l' `pass`instruction pour éviter d'avoir une erreur.

Exemple

```
for x in [0, 1, 2]:
  pass
```