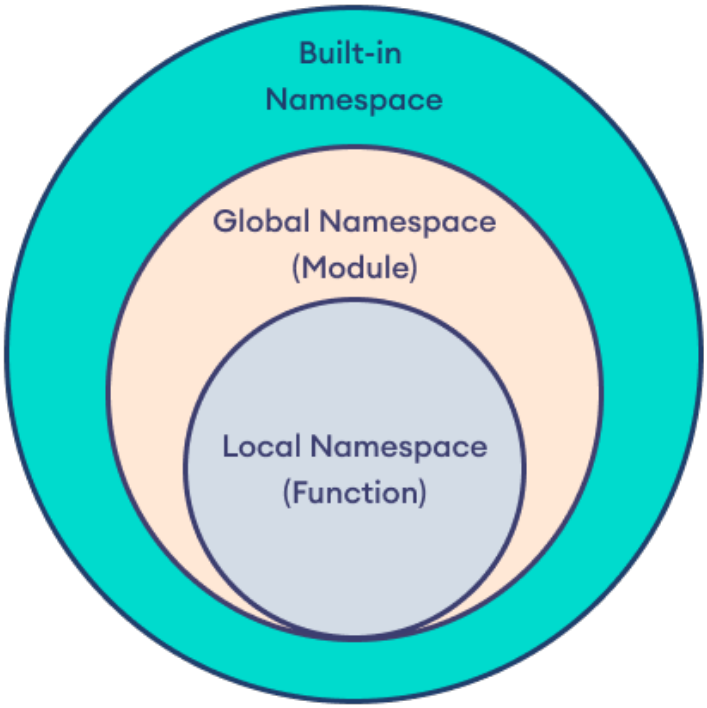
14_python_namespace_and_scope

November 27, 2023

1 Python Namespace and Scope

En python, les variables sont stockées dans des espaces de noms (namespace). Un espace de nom est un dictionnaire qui contient toutes les variables et leurs valeurs. Il existe plusieurs espaces de noms dans un programme python. Par exemple, chaque module a son propre espace de nom. De plus, les fonctions et les classes ont également leur propre espace de nom. Enfin, il existe un espace de nom global qui contient toutes les variables globales. L'espace de nom global est créé lorsque le programme démarre et dure jusqu'à ce que le programme se termine. Lorsque vous utilisez une variable, Python recherche d'abord dans l'espace de nom local, puis dans l'espace de nom global et enfin dans l'espace de nom intégré.

Il est possible d'utiliser plusieurs fois le même nom de variable dans différents espaces de noms. Cependant, les variables ne sont pas liées entre elles. Par exemple, si vous définissez une variable dans une fonction, cette variable n'existe pas dans l'espace de nom global. De plus, si vous définissez une variable dans un module, cette variable n'existe pas dans l'espace de nom d'une fonction ou d'une classe.



Source

si nous avons une fonctions dans une fonction, la fonction interne peut accéder aux variables de la fonction externe, mais pas l'inverse. C'est ce qu'on appelle la portée (scope) des variables. La portée d'une variable est l'endroit où la variable est accessible. La portée d'une variable dépend de l'endroit où la variable est définie. Les variables définies dans une fonction ne sont accessibles que dans cette fonction. Les variables définies dans une classe ne sont accessibles que dans cette classe et les variables définies dans un module ne sont accessibles que dans ce module.

1.1 Exemple 1: Scope and Namespace in Python

[ ]:
```python
# global_var is in the global namespace
global_var = 10

def outer_function(a):
    #  outer_var is in the local namespace
    outer_var = 20
    print("global_var, outer_function", global_var)

    def inner_function():
        #  inner_var is in the nested local namespace
        inner_var = 30
        nonlocal outer_var
        outer_var = 23
        print("global_var, inner_function", global_var)
        print("outer_var, inner_function", outer_var)
```

```
        print("inner_var, inner_function", inner_var)

    inner_function()
    print("outer_var, outer_function", outer_var)

print("global_var", global_var)
# call the outer function and print local and nested local variables
outer_function(20)
print("outer_var", outer_var)
```

Dans l'exemple ci-dessus, Nous avons définit 3 `Namespace` différents: le `global`, le `local de outer function et le`local de inner function'.

De ce fait : - `global_var` - est dans le `global namespace` avec la valeur `10`, cette variable est accessible dans tout le programme. - `outer_val` - est dans le `local namespace` de la fonction `outer_function` avec la valeur `20`, cette variable est accessible dans la fonction `outer_function` et dans la fonction `inner_function`. - `inner_val` - est dans le `local namespace` de la fonction `inner_function` avec la valeur `30`, cette variable est accessible uniquement dans la fonction `inner_function`.

1.2 Exemple 2: Le mot clé `global`

Le mot clé `global` est utilisé pour définir une variable dans l'espace de nom global. Le mot clé `global` est utilisé à l'intérieur d'une fonction uniquement lorsque nous voulons modifier / lire une variable globale à l'intérieur d'une fonction.

[ ]:
```
# define global variable
global_var = 10

def my_function():
    # define local variable
    global local_var
    local_var = 20

    global global_var
    global_var = 30

# print global variable value
print(global_var)

# call the function and modify the global variable
my_function()

# print the modified value of the global variable
print(global_var)
print(local_var)
```

Ici, quand la fonctio `my_function` est appelée, la variable `global_var` est définie dans l'espace de nom global. En utilisant le mot clé `global`, nous pouvons modifier la valeur de `global_var` à

l'intérieur de la fonction. Si on n'utilise pas le mot clé `global`, la variable `global_var` sera une nouvelle variable locale à l'intérieur de la fonction `my_function` avec la valeur `30`. La variable `global_var` globale sera inchangée.

2 Python Variable Scope

En Python nous pouvons définir des variables dans 3 scopes différents: - `local` - à l'intérieur d'une fonction ou d'une classe - `global` - à l'extérieur de toutes les fonctions et classes - `nonlocal` - à l'intérieur d'une fonction, mais disponible dans une fonction imbriquée

Le scope d'une variable definis la region dans laquelle on peut accéder a la variable, par exemple:

```
def add_numbers():
    sum = 5 + 4
```

Ici, la variable `sum` est définie à l'intérieur de la fonction `add_numbers`. Elle est donc accessible uniquement à l'intérieur de cette fonction (local scope). On appelle cette variable une `variable locale`.

2.1 Python Local Variables

Quand on définit une variable à l'intérieur d'une fonction, elle est accessible uniquement à l'intérieur de cette fonction. On appelle cette variable une `variable locale`. Les `variables locales` ne sont pas disponnible en dehors du `local scope` (la fonction).

[ ]:
```
def greet():
    # local variable
    message = 'Hello'
    print('Local', message)

greet()

# try to access message variable
# outside greet() function
print(message)
```

2.2 Python Global Variables

Les `variables globales` sont définies à l'extérieur de toutes les fonctions et classes. On peut accéder à une `variable globale` depuis n'importe où dans le programme.

[ ]:
```
# declare global variable
message = "Hello"

def greet():
    # declare local variable
    print('Local', message)

greet()
```

```
print('Global', message)
```

Cette fois on peut accéder à la variable `message` et on ne reçoit pas l'erreur `NameError`

[ ]:
```
# declare global variable
message = 'Hello'

def greet():
    # declare local variable
    global message
    message = 'World'
    print('Local', message)

greet()
print('Global', message)
```

2.3 Utilisation du mot clé `global`

Ici, si on modifie la valeur de la variable `message` à l'intérieur de la fonction `greet`, la nouvelle valeur de la variable `message` ne sera pas modifiée à l'extérieur de la fonction `greet`.

Si on veut modifier la valeur de la variable `message` à l'intérieur de la fonction `greet` et que cette modification soit visible à l'extérieur de la fonction `greet`, on doit utiliser le mot clé `global` à l'intérieur de la fonction `greet`.

[ ]:
```
# declare global variable
message = 'Hello'

def greet():
    # declare local variable
    global message
    message = 'World'
    print('Local', message)

greet()
print('Global', message)
```

Python Nonlocal Variables

Les `variable nonlocal` sont définies à l'intérieur d'une fonction imbriquée (fonction à l'intérieur d'une fonction). Les `variables nonlocal` sont utilisées dans les fonctions imbriquées afin de modifier les variables définies dans la fonction parent.

[ ]:
```
# outside function
g = "Salut"
def outer():
    message = 'local'

    # nested function
```

```
    def inner():
        # declare nonlocal variable
        nonlocal message
        global g
        g = "Coucou"
        print(message)
        message = 'nonlocal'
        print("inner:", message)
    inner()
    print("outer:", message)

outer()
print(g)
```

Dans le code ci-dessus, nous avons une fonction imbriquée `inner()` et nous utilisons le mot clé `nonlocal` pour définir la variable `message` à l'intérieur de la fonction `inner()`. La fonction `inner()` est définie dans à l'intérieur de la fonction `outer()`

> **Note:** Si on change la valeur de la variable `message` à l'intérieur de la fonction `inner()`, la modification sera visible à l'extérieur, dans la fonction `outer()`.