Локальні і глобальні змінні

Всередині функції можна використовувати значення змінних, оголошених поза цією функцією:

```
def f():
    print(a)

a = 1
f()
```

Тут змінній **a** присвоюється значення 1, і функція **f** друкує це значення, незважаючи на те, що всередині функції **f** ця змінна не ініціалізується. Але в момент виклику функції **f** змінній **a** вже присвоєно значення, тому функція **f** може його використовувати.

Такі змінні (оголошені поза функцією, але доступні всередині функції) називаються глобальними.

Якщо ініціалізувати якусь змінну всередині функції, використовувати цю змінну поза функцією не вдасться. Наприклад:

```
def f():
    a = 1

f()
print(a)
```

Отримаємо NameError: name 'a' is not defined. Такі змінні, оголошені всередині функції, називаються локальними. Ці змінні стають недоступними після виходу з функції.

Цікавим вийде результат, якщо спробувати змінити значення глобальної змінної всередині функції:

```
def f():
    a = 1
    print(a)

a = 0
f()
print(a)
```

Будуть виведені числа 1 і 0. Тобто незважаючи на те, що значення змінної **a** змінилося всередині функції, то поза функцією воно залишилося незмінним. Це зроблено з метою "захисту" глобальних змінних від випадкової зміни з функцій. Тобто, якщо всередині функції модифікується значення деякої змінної, то змінна з таким ім'ям стає локальної змінної, і її модифікація не приведе до зміни глобальної змінної з таким же ім'ям.

Більш формально: інтерпретатор Python вважає змінну локальною, якщо всередині неї є хоча б одна інструкція, що модифікує значення змінної (це може бути оператор =, + = і т.д.). Або при використанні цієї змінної в якості параметра циклу for, ця змінна вважається локальною. При цьому навіть якщо інструкція, що модифікує змінну ніколи не буде виконана: інтерпретатор це перевірити не може, і змінна все одно буде вважатися локальної. Приклад:

```
def f():
    print(a)
    if False:
        a = 0

a = 1
f()
```

Виникає помилка: UnboundLocalError: local variable 'a' referenced before assignment. А саме, у функції **f** ідентифікатор **a** стає локальної змінної, тому що в функції є команда, що модифікує змінну **a,** а саме (a = 0), нехай навіть ніколи і не виконується (інтерпретатор не може це відстежити). Тому вивід змінної **a** призводить до звернення до неініціалізованої локальної змінної.

Щоб функція могла змінити значення глобальної змінної, необхідно оголосити цю змінну всередині функції, як глобальну, за допомогою ключового слова global:

```
def f():
    global a
    a = 1
    print(a)

a = 0
f ()
print (a)
```

У цьому прикладі на екран буде виведено двічі одиницю, так як змінна **a** оголошена, як глобальна, і її зміна всередині функції призводить до того, що і всередині функції і поза її межами, змінна буде доступна.

Проте, краще не змінювати значення глобальних змінних усередині функції. Якщо функція повинна змінити значення певної змінної, то як правило це краще оформити як значення, що повертається функцією.

Якщо потрібно, щоб функція повернула не одне значення, а два або більше, то для цього функція може повернути кортеж з двох або кількох значень:

```
return (a, b)
```

Тоді результат виклику функції теж потрібно присвоювати кортежу:

```
(n, m) = f (a, b)
```