Scope

Introduction to Computer Programming

Dr. Paul Vrbik

September 18, 2018

From last time — this is totally wrong

Question

Trace the following.

```
>>> 3>=3 and 7>7 or 1>0
True
>>> (3>=3 and 7>7) or 1>0
True
>>> 3>=3 and (7>7 or 1>0)
True
```

~~*This means that* `and` *and* `or` *are associative operations.*~~

~~That is, you can bracket in any order.~~

```
>>> 3>3 and 7>7 or 1>0
True
```
according to Python

```
>>> 3>3 and (7>7 or 1>0)
False
```

```
>>> (3>3 and 7>7) or 1>0
True
```

Example

Consider that

```
True or False and False == True
```

is ambiguous because

```
(True or False) and False == False
True or (False and False) == True.
```

and thus an order of operations is necessary to resolve ambiguities.

Definition (Oder of operations)

In the case of ambiguity `and` is evaluated before `or` .

The `or` and `and` operations are individually associative but are not associative when mixed.

Back to regular scheduled programming... HA!

Warmup

```
>>> def foo():
...     return 9
>>> foo()
9
>>> def bar(x):
...     x = 8
...     return x
>>> bar(1)
8
>>> bar(2)
8
```

```
>>> def square(x):
...     print(x**2)

>>> a = square(2)
4

>>> a == 4
False
>>> a == None
True
```

Do not print your answer — return it. We are asking you to design functions and not a user interface.

```
>>> def foo(x):
...     return x + 2
...     return x + 3

>>> foo(10)
12
>>> ans = foo(10)
>>> ans
12
```

Everything after the first return statement is ignored.

```
>>> def foo(x):
...     print(x + 2)
...     print(x + 3)

>>> foo(10)
12
13
>>> ans = foo(10)
12
13
>>> ans
None
```

Definition (Scope)

Suppose a computer program creates a variable.

The scope of that variable is the collection of places (e.g. functions, procedures, control structures) that can access its value.

```
>>> x = 2
>>> y = 3
>>> def foo():
...     return x

>>> def bar():
...     return foo()*y

>>> foo()
2
>>> bar()
6
```

(x and y are global variables available to all functions.)

Definition (Global Variable)

A global variable (or simply 'global') is one that can be accessed by all functions.

Anything declared outside a function will be globally accessible.

A function declared globally is said to have global scope.

Constants

By convention constants are defined in caps

```
PI = 3.14159

NUMBER_OF_DAYS_IN_WEEK = 7
```

```
>>> x = 2


>>> def foo():
...     x = 7
...     return


>>> foo()
>>> x
2
```

Despite having the same name, the `x` of `foo()` is assumed local — its scope is `foo()`.

```
>>> x = 2
>>> def foo():
...     global x
...     x = 7
...     return

>>> foo()
>>> x
7
```

We can specify that `foo` should be using `x` as a global. It is good practice to declare your globals when you use one.

```
>>> def foo():
...     x = 2
...     return x

>>> foo()
2

>>> x
NameError: name 'x' is not defined
```

Outside of `foo` the variable `x` does not exist; `x` is a local variable.

```
>>> x = 2
>>> def foo():
...     x = x + 2
...     return

>>> foo()
UnboundLocalError: local variable x'
referenced before assignment.
```

When `foo` creates `x` it becomes local and thereby has no value at the time of assignment.

```
>>> x = 2
>>> def foo():
...     global x
...     x = x + 2
...     return

>>> foo()
>>> x
4
>>> foo()
>>> x
6
```

```
>>> x = 5
>>> def foo(x):
...     return x

>>> foo(7)
7

>>> x
5
```

Despite having the same name there are two `x`'s: one with a global scope and another with local.

```
>>> x = 5
>>> def foo(y):
...     return x*y

>>> foo(7)
35

>>> foo(x)
25
```

Globals and locals can be used in mixed computation.

```
>>> x = 5
>>> def foo(x):
...     global x
...     return
SyntaxError: name ‘x’ is parameter and global
```

Design Recipe

Step 1

Pick a short, descriptive, name for the function. A good name answers the question "What does your function do?"

Design Recipe

Step 2

Write your function header with a docstring. Assume your function works already and give examples of how to use it.

```
def is_prime(x: int) -> bool:
    """
    >>> is_prime(7)
    True
    >>> is_prime(8)
    False
    """
```

Design Recipe

Step 3

Write a short and concise description of your function.

```
def is_prime(x: int) -> bool:
    """ Return True only when x is a prime
    >>> is_prime(7)
    True
    >>> is_prime(8)
    False
    """
```

Step 4

Write your function. Return your answer.

```
def is_prime(x: int) -> bool:
    """ Return True only when x is a prime
    >>> is_prime(7)
    True
    >>> is_prime(8)
    False
    """
    .
    .
    return answer
```

Step 5

Test your function. Be sure to include corner cases.

Next Time

1. Writing functions.