Introduction to Programming in Python

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

More on functions

Docstrings

Python allows inline documentation via *docstrings*. This is just a string that appears directly after the function definition and documents what the function does:

In [ ]:
```
def mypower(x, y):
    """Compute x^y."""
    return x**y
```

- It is customary to use a triple quoted string; these can contain newlines.
- The docstring is shown by the help function

In [ ]:
```
help(mypower)
```

This explains the difference between a comment and a docstring: the former is for the developer, the latter for the user.

Exercise, continued

Add a docstring to the `area` function from last week:

In [ ]:
```
def area(a, b=None):
    if b:
        return a * b
    else:
        return a * a
```

Variable Scope

- Variables defined in functions are local (not visible in the calling scope):

In [ ]:
```
def f():
    z = 1
f()
```

In [ ]:
```
print(z)
```

- The same is true of the input arguments:

In [ ]:
```
def f(num):
    return num**2
```

In [ ]:
```
num
```

Variables defined outside of functions are `global` : they are visible everywhere:

In [ ]:
```
a = 3
def f():
    print(a)
```

In [ ]:
```
f()
```

That is, unless they area "shadowed" by a local variable:

In [ ]:
```
a = 3
def f():
    a = 2
    print(a)
```

In [ ]:
```
f()
print(a)
```

The `global` statement

If we do actually want to act upon the global variable, then we need to be explicit about it:

In [ ]:
```
a = 3
def f():
    global a
    a = 2
    print(a)
```

In [ ]:
```
f()
print(a)
```

Quiz

For each of the following, state what gets printed.

1.

```
def f():
    name = "Alexander"
name = "Simon"
f()
print(name)
```

2.

```
def f():
    global name
    name = "Alexander"
name = "Simon"
f()
print(name)
```

3.

```
def f():
    global name
    name = "Alexander"
name = "Simon"
print(name)
```

4.

```
def f(x):
    x = x + 2
x = 7
f(x)
print(x)
```

5.

```
def f(x):
    x[0] = x[0] + 2
    return x
y = [7]
f(y)
print(y[0])
```

Mutating functions

- That last example was a bit of a curveball.
- Turns out that if you pass a mutable argument (like a `list`) into a function, then changes to that variable are visible to the caller (i.e., outside the function):

In [ ]:
```
def f(y):
    y[0] = 2
```

In [ ]:
```
x = [1]
f(x)
print(x)
```

Splatting and Slurping

- Splatting: passing the elements of a sequence into a function as positional arguments, one by one.

In [ ]:
```
def mypower(x, y):
    return x**y
args = [2, 3]  # a list or a tuple
mypower(*args)  # splat (unpack) args into mypower as positional arguments.
```

- Slurping allows us to create *vararg* functions: functions that can be called with any number of positional and/or keyword arguments.

In [ ]:
```
def myfunc(*myargs):
    for i in range(len(myargs)):
        print("Argument number " + str(i+1) + " was " + str(myargs[i]) + ".")
```

In [ ]:
```
myfunc(3, 5)
```

- I.e., The asterisk means "collect all (remaining) positional arguments into the tuple `myargs`".
- This is essentially how the built-in `print` function works.

Recap / further reading (optional)

- https://www.w3schools.com/python/python_functions.asp (https://www.w3schools.com/python/python_functions.asp)
- https://python-course.eu/python-tutorial/functions.php (https://python-course.eu/python-tutorial/functions.php)