Functions

Simple Functions

Definition:

```
def function_name():
    ... code goes here ...
```

Calling:

```
function_name()
```

Functions can only be called after they have been declared!

Simple Functions

```
def greet():
    print(“Well, hello there!”)

greet()
```

Functions With Parameters

Definition:

```
def function_name(param1, param2, etc...):
    ... code goes here ...
```

Calling:

```
function_name(value1, value2, etc...)
```

Functions With Parameters

```
def greet(name):
    print("Hello, %s!" % (name))

greet("Alex")
```

Exercise: Sum and Product Function

Write a function that takes two numbers, and prints their sum and their product. Call this function several times with different values.

Exercise: Sum and Product Function

Write a function that takes two numbers, and prints their sum and their product. Call this function several times with different values.

```
def sum_and_product(x, y):
    print("Sum: %d" % (x + y))
    print("Product: %d" % (x * y))

sum_and_product(2, 3)
sum_and_product(8, 12)
sum_and_product(10, 10)
```

Returning a Value

Terminate a function with:

```
return
```

Terminate a function, returning a value, with:

```
return some_value
```

Returning a Value

```
def myfunction():
    print(“Hi there!”)
    return
    print(“This message will never be shown...”)

myfunction()
```

Returning a Value

```
def add(x, y):
    return x + y


z = add(3, 4)
print(z)          # Outputs 7
```

Exercise: Circle Properties

```
r = float(input("Radius: "))
pi = 3.14159
circ = 2 * pi * r
area = pi * r * r
print("Circumference: %f" % circ)
print("Area: %f" % area)
```

Rewrite this program to use two functions: one to calculate and return the circumference, and one to calculate and return the area. Take an input radius, pass it to both functions, and print the values returned.

Exercise: Circle Properties

```
pi = 3.15159

def circ(r):
    return 2 * pi * r

def area(r):
    return pi * r * r

r = float(input("Radius: "))
print("Circumference: %f" % circ(r))
print("Area: %f" % area(r))
```

Returning Multiple Values

We can return any data type we want! To return multiple values, just bundle them up in a tuple (or list, or other compound data structure).

```
def sum_and_product(x, y):
    return (x + y, x * y)

(s, p) = sum_and_product(6, 7)
# s = 13
# p = 42
```

Variable Scope

Scope: What part of the program a variable exists in.

Global scope: Accessible throughout the program.

Variables defined in the main program have global scope.

Local scope: Accessible only in a particular function.

Variables defined in a function have scope local to that function.

Local variables have priority over global variables if their names clash.

Variable Scope

```
pi = 3.15159

def circ(r):
    return 2 * pi * r

def area(r):
    return pi * r * r

r = float(input("Radius: "))
print("Circumference: %f" % circ(r))
print("Area: %f" % area(r))
```

Default Parameters

```
def greet(name = “User”):
    print(“Hello, %s!” % (name))

greet(“Alex”)        # “Hello, Alex!”
greet()              # “Hello, User!”
```

Default Parameters

```
def greet(name = "User"):
    print("Hello, %s!" % (name))

greet("Alex")        # "Hello, Alex!"
greet()              # "Hello, User!"
```

Default parameters must come at the end:

```
def good(a, b = 2)
def bad(a = 6, b)
```

Specify Parameters by Name

```
def greet(greeting, name):
    print("%s, %s!" % (greeting, name))
```

Normally, we might call:

```
greet("Hello", "Alex")
```

But we could also use:

```
greet(name = "Alex", greeting = "Hello")
```

Varying Number of Parameters

```
def many_args(param1, param2, *otherparams):
    ... some code ...
```

Calling:

```
many_args(4, “hello”, “x”, “y”, “z”)
```

Will set the parameters of `many_args` to:

```
param1 = 4
param2 = “hello”
otherparams = (“x”, “y”, “z”)
```

Passing by Reference - Watch Out!

In Python, values are passed to function parameters '*by reference*'.

This means that, even though the variables in your function are different to those you might have passed in, they both refer to the *same memory*.

This shouldn't be much of a problem until we cover classes and objects in a future lesson.

Passing by Reference - Watch Out!

```
def myfunction(y):
    y = 5

x = 2
print(x)          # x is 2
myfunction(x)
print(x)          # x is 2
```

```
def myfunction(y):
    y.append(5)

x = [2, 4]
print(x)          # x is [2, 4]
myfunction(x)
print(x)          # x is [2, 4, 5]
```

Recursion

A function can call another function:

```
def myfunction():
    print(“Hi!”)

def otherfunction():
    myfunction()
    print(“Other function”)
```

So naturally, a function could also call itself!

Recursion

We should avoid looping forever…

```
def badfunction():
    print(“Let’s go on forever!”)
    badfunction()
```

To be useful, a recursive function will need a terminating condition.

Recursion

Calculating integer powers (really inefficiently!):

```
def power(x, n):
    if n == 1:
        return x
    else:
        return x * power(x, n - 1)
```

Exercise: Factorial

The *factorial* (!) of a number is defined as the product of every integer between 1 and that number.

e.g. 5! = 1 * 2 * 3 * 4 * 5 = 120

Write a recursive function find the factorial of an input.

Exercise: Factorial

```
def factorial(x):
    if x == 1:
        return x
    else:
        return x * factorial(x - 1)
```