Review

Everything in python is an object that has an associated type. You can always find out the type of an object by using the `type` function. There are many builtin types in Python. We've already seen `int`, `float`, and `str`. There is a boolean type `bool` that has only two values `True` and `False`. There is also a `NoneType` called `None`, which we will talk about more later.

Recall the built-in functions below:

- `int(x)` : construct an integer from some object `x`, which is usually has type integer, float, or string.
- `str(x)` : construct a string representation of some object `x`.
- `float(x)`: construct a float from some object `x`, which is usually has type integer, float, or string.

Modules and Packages

A *module* is a file containing Python code—function definitions, expressions, and statements. Modules provide a way to logically group collections of related code, keep the global namespace clear, and distribute code to others. By using the statement

```
import mymodule
```

we are letting python know that there is a file called `mymodule.py` somewhere in the Python search path that contains functions about which we care. We can now access these functions using the *dot notation* `my module.function_name`.

Python Script Mode

Let's write a program called `sum.py` that takes two arguments from the command line and prints out their sum.

```
import sys

x = sys.argv[1]
y = sys.argv[2]

print("The sum of " + x + " and " + y + " is " + (x+y))
```

First some explanation. The module `sys` gives us access to a variable called `argv`, which is a vector of strings that appear on the command line. `sys.argv[0]` is the name of the script. `sys.argv[1]` is the first argument, `sys.argv[2]` is the second argument, and so on. Let's run this script in script mode.

```
$ python3 sum.py 5 6
The sum of 5 and 6 is 56
```

Um, that's not right. What's wrong? The arguments are strings of characters, not numbers.

```
import sys

x = int(sys.argv[1])
y = int(sys.argv[2])

print("The sum of " + str(x) + " and " + str(y) + " is " + str(x+y))
```

1 Functions

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

def Use `def` to define a function. The syntax is

```
def function_name(param1, ..., paramk):
```

parameters a function definition involves both a *name* and a set of (perhaps zero) *formal parameters*. The formal parameters are named and act as local variables.

function header The entire first line of the function is called the *function header*

docstring The string following the function header is called the *docstring*. This string gets bound to the `__doc__` method associated with the function object

function body The body of a function is a sequence of python expressions. Notice that indentation level determines to which block the code is associated.

variables The variables defined within a function are local to that function. Their scope extends to blocks of code at or beneath the same level, but above that level. This is called lexical scoping—resolution of variables depends on where the variables are defined.

mathematical functions versus procedures The `return` statement can be used to return a value from a function. Functions that return values are *mathematical functions* whereas functions that don't return values are *procedural functions*.

function objects Everything in Python is an object, including functions. This means that functions can be passed as arguments to other functions, which can then apply them to other arguments. Languages with this ability are said to support *higher-order functions*.

```
def polar(x, y, deg=False):
    '''convert (x,y) into polar coordinates
       where the angle is in radians (default)
       or degrees (deg=True)
    ??'
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    if deg:
        return (radius, angle * 180 / math.pi)
    else:
        return (radius, angle)
```

default values Arguments may have default values; arguments without default values cannot appear after arguments with default values.

conditionals Conditional statements allow you to branch the flow of execution.

indentation of conditionals The control flow of conditional statements follows the rules of indentation

equality test The test of a conditional statement has type `bool`. Every object in Python has a boolean value: False, None, 0, and empty data structures evaluate to False; all others evaluate to True. Use the equality operator (==) to compare values.

2 Group Exercises

Write python code to compute the following functions:

- `even(x)`: returns *True* if and only if `x` is even
- `odd(x)`: returns *True* if and only if `x` is odd
- `min(x,y)`: returns the smaller of `x` and `y`
- `max(x,y)`: returns the larger of `x` and `y`
- `perfect_square(x)`: returns `True` if and only if `x` is a perfect square (i.e. its square root is an integer)
- `fact(x)`: returns `x!` (note: `fact(0) = 1` and `fact(n) = n * fact(n-1)`