References ("ref:") are from
http://docs.python.org/py3k/reference/

Values and Variables

Variables are not declared;
Variables can be assigned any type of value at any time using =.

```
average = ( first + second ) / 2
```

1b. Operators

add **+** ; subtract **–** ; multiply ∗ ; power ∗∗
- Truncating (round-down) division: `//`
- Normal division: `/`
- String concatenation uses **+**

Comparison (**==**,**!=**,**<**,**<=**,**>**,**>=**) checks object content, (not *addresses*) for all standard types.

Logic operators: **and**, **or**, **not**

String - type name "str"

Use double or single quotes.
There is no separate character type.
To make a multi-line string, **"""use 3 double (or single) quotes."""**
Indexing with brackets (**s[i]**) works.

"if' statements

```
if x < y:
      print(“Big!”)
else:
      print(“small.”)
      x = x + 1
```


Common functions

```
int( “52” ) # The integer 52
int( 98.6 ) # The integer 98
str( 52 ) # The string “52”
float( 52 ) # The float 52.0

x = 42
y = 24
print( x )
# Prints 42 on its own line

print( x, y )
# Prints “42 24” on one line

print( str(x) + “|” + str(y) )
# Prints “42|24”

n = int( \
      input( “Number, please: ” ))
# Reads in literal string;
# int() converts it
```

Importing Packages

To use code from another Python file...

```
import math

if x >= 0:
      print( "Square root of", \
             x, " is ", \
             math.sqrt( x ) )
else:
      print( "Negative number!" )
```

Alternative (beware of name conflicts.)

```
from math import ∗

if x >= 0:
      print( "Square root of", \
             x, " is ", sqrt( x ) )
else:
      print( "Negative number!" )
```


Defining Your Own Functions

ref: compound_stmts.html#function-definitions

```
def order( val1, val2 ):
    """State which value naturally
       comes first.
    """
    if val1 < val2:
        print(val1, “comes first”)
    else:
        print(val2, “comes first”)


def sum3( a, b, c ):
    “Add 3 numbers.”
    return a + b + c
```

The string that follows the header is used for documentation generation.

```
order( "joe", "black" )
# Prints “black comes first”

order( 13, 21 )
# Prints “13 comes first”


print( sum3( 1, 5, 9 ) )
# Prints 15
```

For loops

```
for n in [ "how", "are", "you" ]:
      print(n)
# Prints "how", "are", and "you"

for n in range( 5 ):
      print(n)
# Prints 0, 1, 2, 3, and 4

for n in range( 10, 0, -2 ):
      print(n)
# Prints 10, 8, 6, 4, and 2
```

While loops

```
n = 10
while n > 0:
      print(n)
      n = n - 2
# Prints 10, 8, 6, 4, and 2
```

More about Data Model

Everything in Python is an object. Assignment (=) effects sharing of data.

```
x = [ 1, 2, 3 ] # a list
y = x
x[ 1 ] = 5 # 2 changed to 5
print(y) # prints "[1, 5, 3]"
```

Numbers (**float**, **int**), **bool**s, and **str**ings can't be changed; they are for all intents and purposes not shared.

None is used for a variable with no value.

An *immutable* object cannot have its contents changed. (But a variable referring to an immutable object can be reassigned to a new object.)

Built-in data structures

ref: datamodel.html#the-standard-type-hierarchy

All of the following can be iterated over with a **for** loop.

String (immutable) – str

(See reverse side.)

List (mutable; see 1a) – list

```
x = ["r","o","o","f"]
# works with the str "roof" as well
# Example of using an index
for i in range( len( x ) ):
      print(x[i])
# Prints "r", "o", "o", and "f"
```

Tuple: an immutable list – tuple

```
y = ( 4, 5, 6 ) # can't be changed
```

Dictionary/Set (mutable) – dict/set

```
d = { "fee": 9, "fo": 18 }
# Order of keys is not settable.
d["fum"] = 21
d["fo"] = 17
for key in ("fum","fee","fo"):
      print(d[key])
# Prints 21, 9, and 17
```

A **set** is just a **dict** containing keys without values.

```
names = {"Manny","Moe","Jack"}
```


Defining Your Own Classes

Use a *class* to define your own composite data type.

Sample Class Definition

ref: compound_stmts.html#class

```
class Point( object ):
    "A 2-dimensional point"
    __slots__ = ( "x", "y" )

    def __init__( self, x, y ):
        "constructor"
        self.x = x
        self.y = y
    def distFromOrigin( self ):
        return \
          math.sqrt( self.x**2 + \
                     self.y**2 )
    def __str__( self ):
        "to-string converter"
        return "(" + \
               str( self.x ) + \
               "," + \
               str( self.y ) + ")"
```

Examples of Class Use

```
def test():
    p = Point( 3, 4 )
    print(p.x)
    print(p)
    print(p.distFromOrigin())

test()
# Prints 3, "(3,4)", and 5.0
```