Control Flow

Instead of operators like in Java (&& ||)
you just use the words (and, or, not)

Everything is technically public, but convention says that private has two underscores, protected is implied with a single underscore

Functions

Define the function

def function_name(myVariable, myOtherVariable = 10):

call the function

function_name(myVariable)

Multiple Returns

You can return multiple things

return myInt1 myInt2 myInt3

a, b, c = myIntFunction()

If/elif/else statements

"if" followed by a statement followed by a ":"
if {statement} :
result

Range

range(10) = Creates an **object** that contains the values 0 to 10
We can make this into a list by using list(range(10))

range(0,5) Creates an object that contains the values 5 to 10

range(0,100,10) Creates an object that contains 0,10,20,30,40,...,100

For Loops

for myObject in myList:
print(myObject)

Or

for i in range(6):
print(i)

Or

for i in range(len(myList))

While Loops

while myVariable <=3:
count += 1

Or

length = len(myList)
index = 0
while index < length:

Try/Except

Variable_to_check = "myVariable"
try:
print(myDict[Variable_to_check])
except:
print("That key doesn't exist!")

Operators

a *b c = (1, 2, 3, 4, 5)

would give b a value of (2,3,4)

Scope

Keyword: nonlocal

nonlocal myVariable ... accesses a variable from the enclosing scope, rather than creating a new variable in the local scope

Keyword: global

global myVariable ... allows you to modify a global variable, (rather than just access it)

Lambda

```
def add_two(my_input):
  return my_input + 2
```

can also be written in lambda format

```
add_two = lambda my_input: my_input + 2
```

```
print(add_two(3))
print(add_two(100))
print(add_two(-2))
```