PYTHON

-Functions:

Defined with 3 components: the header, which includes the **def** keyword, the name of the function, and any parameters the function requires; an optional comment enclose in triple quotes(“””); and the body, which describes the procedures the function carries out.

A function can have any number of **parameters**, a variable that is an input to a function. The values of the parameters passed into a function are known as the **arguments**. To call a function you should pass in the same number of arguments as there are parameters.

def new\_function(parameter): #a parameter being any input

new\_function(argument) #calls the function with the parameter set to an argument

def function\_two(x): #the first calls another function

return function\_one(x) + n

max(argument, argument, argument) #returns the largest number

min(argument, argument, argument) #returns the smallest number

abs(argument) #returns the absolute of a number

type(argument) #returns the type (int, float or str)

-Lambda:

An anonymus function. When passed to **filter**, uses the lambda to determine what to filter, and the second argument, is the list it does the filtering on.

Lambda x: code

filter(lambda x:code, list\_name)

-Lists:

Use to store different pieces of information (items) as a sequence under a single variable name.

Each item has an **index** starting at 0.  It can be used to access as well as assign values. An index **can be a code** that defines a number.

It is possible to **slice** a string like a list, each character is a sequential item starting at index 0.

A **for** loop will do something with every item in the list. The indented code that follows it will be executed once per item in the list.

The blocks of code in a **for** loop can be as big or as small as they need to be.

A **range** generates a list of numbers from start up to (but not including) stop. Each item increses by step. Start defaults to 0 and step to 1.

list\_name = [item, item]

empty\_list = []

list\_name[index] #gets the value at index

list\_name[index] = item #changes de value

list\_name.append(item) #add item to the list at the end

list\_slice[index:index] #slice a portion including the first index but excluding the last one

list\_slice[:index] #from the start

list\_slice[index:] #to the end

list\_slice[index:index:stride] #the stride is an index that defines the space between the items considered, a

negative one traverses the list from right to left .

list\_name.index(item) #gets the index number of the item

list\_name.insert(index, item) #insert an item at the index

list\_name.remove(item) #remove an item

del(list\_name[index]) #remove an item

list\_name.pop(index) #remove and return an item

for variable in list\_name: #variable can be set to any name except reserved words

list\_name.sort() #modifies the list in alphabetical order

range(stop)

range(start, stop)

range(start, stop, step)

“#element ”.join(list\_name) #removes the “” when printing a string and place #element in between

string.split(separetor) #returns a list of all the words in the string using a defined separator

(default whitespace)

enumerate.(list\_name) #supplies a corresponding index to each element in the list; each time you go

through the loop, **index** will be one greater, and **item** will be the next item in

the sequence.

-Dictionary:

Similar to a list, but access values by looking up a key that can be any string or number.

A value can be a list.

The lenght of a dictionary is the number of key-value pairs it has. Each pair counts only once, even if the value is a list.

It is possible to use a **for** loop on a dictionary. Any time you loop through it, you will go through every key, but you are not guaranteed to get them in any particular order since dictionarys are unordered.

dictionary = {key : value, key : value, key : [“a, “b”], key : value} #key has a list as a value

empty\_dictionary = {}

dictionary[“key”] #gets the value

dictionary[key] = value #add a key-value pair or change de value of that key

del dictionary[key] #remove the key and its value

dictionary.items() #return key/value in any order

dictionary.keys() #return a list of the keys

dictionary.values() #return a list of the values

dictionary[“key\_with\_list”].commandlist #execute a command to the list in the value

for key in dictionary: #

-If:

**If** executes the code inside of it if some condition is true. **Elif** allows you to check multiple expressions. **Else** contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

if example:

code

elif example:

code

else:

code

-While:

Similar to an if statement, but will continue to execute as long as the condition is true. The condition is the expression that decides whether the loop is going to continue being executed or not. The **else** block will execute **anytime** the loop condition is evaluated to false.

while condition:

-Break:

The **break** statement terminates the current loop and resumes execution at the next statement.

break

-For:

This statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

for #example in #example if #example

list = [x for x in range() if]

-Bitwise operations:

Operations that directly manipulate bits. In all computers, numbers are represented with bits, a series of zeros and ones.

10 = 0b1010

bin(int) #returns the int binary representation

oct(int) #returns the int base 6 representation

hex(int) #returns the int base 8 representation

int(“number”, base\_of\_number) #returns the value of that number converted to base ten