Python notes:

Variable Types:

Float is a decimal: var = 7.5

Int is an integer: var = 7

Bool is a true or false: var = True

Operators:

Number operators	Bool operators	Functional operators
+ addition	== equal to	And
- Subtraction	!= not equal to	Not
* multiplication	>= greater than or equal to x	Or
** exponent	<= less than or equal to x	
% modulo	< greater, > less	

Statements and python functions:

Print(): prints “str” or var or (“str”, var) or (“str” + “str”)

Return(): returns an equation or argument

Comments: # is used to make python comments

Range(a, b, c): gives numbers from a to b with a difference of c between each number

.upper(): capitilises a whole string

.lower(): decapitalises a whole string

.title(): capitalises the first letter of each word in a string

.split(x): splits each word or section at character ‘x’ in a string into a list

.join(): used as “ variable= ' '.join(list)“ turns a list of words into a sentence or string joined by ‘ ‘

.find(): “string.find(‘element’) returns index of start of element

.strip(): ‘ ‘.strip(list) returns string of list elements stripped of ‘ ‘

.replace(): new_string = string.replace(‘a’, ‘b’) replaces all ‘a’ with ‘b’ and saves to a new string

.format(): “blah {paraeter1} blah blah {parameter2”.format(parameter1, parameter2) allows you to interpolate a string with variables

.split(: String.split(‘thing to split by’) splits string into a list using the entered thing to split by

.now():

.plot(a, b): plots a graph of a against b (pyplot function)

.show(): var.show() returns visual of var to browser (pyplot function)

Random python functions: (must import random library first)

random.randint(a, b): produces a random integer between a and b

random.choice(var): returns a random index of var

random.sample(sequence, k): returns k length list of elements chosen from sequence

Defining functions:

Def function(parameters):

Function arguments.

Strings:

Strings are stored as text between quotes “text” or ‘text’

Parts os strings can be called or used by other variables using index notation

e.g. name = “Evey”

```
age = “25”
user_name = name[:2]+age

print(user_name)
```

will print:

Ev25

Adding quotes in the middle of a string:

```
String = “I am “Evey” May”  #this wont work
String = “I am /”Evey”/ May”  #using escape characters will allow this
```

All string code:

```
def username_generator(first_name, last_name):

    if len(first_name) < 3:
        username = first_name+ last_name[:4]
    elif len(last_name) < 4:
        username = first_name[:3] + last_name
    else:
        username = first_name[:3] + last_name[:4]
    return username

def password_generator(username):
    password = ""
    for index in range(0, len(username)):
        password += username[index-1]
    return password
```

Lists:

My_list = [items separated by commas]
My_list.append(item) (adds item to end of the list)
My_list2 = My_list + [1, 3] (adds 1 and 3 to the end of My_list)
My_list3 = zip(list1, list2) (makes a list of lists [[list1a, list2a], [list1b, list2b]]
Var = len(my_list) (gives amount of items in the list)
A chunk of list elements can be assigned to a new variable using index:

```
Elementx = list[x]
Start = list[0:x] or list[:x]
Middle = list[x:y]
End = list[x:y] or list[x:]
```

List.sort() sorts list in alphabetical order
Sorted(list) takes the alphabetically sorted list and assigns it a value
List.Count(item) counts the amount of occurrences of item in List

Tuples:

Very similar to lists however are immutable and cannot be added to, changed or edited once defined.
My_tuple = (items)
If creating a one element tuple, a comma must go after the assignment (e.g. one_element_tuple = (‘element’,)
Assigning variables in (or unpacking) a tuple:

```
My_tuple = (“evey”, 25, “tutor”)
```

Name, age, job = my_tuple
This will assign name = “evey”, age = 25 and job = “tutor”

Loops:

In lists a *for loop* can be used to print each list item without the constraints of its list:

e.g. girls_names = [Emma, May, Bella, June]

```
for names in girls_names:
    print(names)
```

console will print:

Emma
May
Bella
June

For loops may also be used in conjunction with range() to print multiple lines of the same thing.

```
promise = "I will not chew gum in class"
for i in range(5):
    print(promise)
```

console will print:

I will not chew gum in class
I will not chew gum in class
I will not chew gum in class
I will not chew gum in class
I will not chew gum in class

Adding lists together:

```
students_period_A = ["Alex", "Briana", "Cheri", "Daniele"]
students_period_B = ["Dora", "Minerva", "Alexa", "Obie"]

for i in students_period_A:
    students_period_B.append(i)
```

If statements in for loops:

e.g. 1

```
dog_breeds_available_for_adoption = ['french_bulldog', 'dalmatian', 'shihtzu
', 'poodle', 'collie']
dog_breed_I_want = 'dalmatian'

for breed in dog_breeds_available_for_adoption:
    print(breed)
    if breed == dog_breed_I_want:
        break #break stops the loop once the breed is found

print('They have the dog I want!')
```

e.g.2

```
ages = [12, 38, 34, 26, 21, 19, 67, 41, 17]

for age in ages:
    if age >= 21:
```

```
print(age)
```

For loops inside for loops:

```
sales_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]
scoops_sold = 0

for locations in sales_data:
    for scoops in locations:
    scoops_sold += scoops #adds each number to the scoops_sold variable

print(scoops_sold)
```

While loops:

```
all_students = ["Alex", "Briana", "Cheri", "Daniele", "Dora", "Minerva", "Alex
a", "Obie", "Arius", "Loki"]
students_in_poetry = []

students = 0
while len(students_in_poetry) < 6:
    students = all_students.pop()  #.pop() takes item from the end of the list
    students_in_poetry.append(students)

print(students_in_poetry)
```

List comprehension:

While we can write an entire loop and append to a new list such as:

```
words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply",
"timestamp", "@matchamom", "follow", "#updog"]
usernames = []

for word in words:
    if word[0] == '@':
    usernames.append(word)
```

We can also use list comprehension, which is a single line piece of code that does the same thing.

```
Usernames = [word for word in words if word[0] == '@']
```

Another example:

```
celsius = [0, 10, 15, 32, -5, 27, 3]
fahrenheit = [temp * 9/5 + 32 for temp in celsius]

print(fahrenheit)
```

Putting it all together:

```
# regular for loop method
single_digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
squares = []

for digits in single_digits:
    print(digits)
    squares.append(digits**2)
print(squares)

# list comprehension method
cubes = [digits**3 for digits in single_digits]
print(cubes)
```

Dictionaries:

Creating dictionaries:

formatting: dictionary = {"key": "value", "key2":"value2", "key3":"value3"}
Value can also be a list whereas keys must be a string.

Adding to existing dictionary:
Dictionary.update({"newkey" : "newvalue", "newkey2" : "newvalue2"})

Changing dictionary values:
Dictionary["key"] = "new value"

Using zip and list comprehension in dictionaries:
dictionary = {key: value for [key, value] in zip(list1, list2)}

Using dictionaries:

Accessing specific values:
Print(Dictionary["key"]) will print the value/values for the key

Try/ except block:

```
Dictionary = {key: value}
Try:
    Print(dictionary[key])
Except KeyError:
    Print("no values for this key")
```

.get():

```
Dictionary = {key: value}
Dictionary.get("key", default) #default is an optional parameter
```

.pop():
Adding elements to a variable from a dictionary.

```
Dictionary = {key: value}
Variable = int
Variable += dictionary.pop(key, alternate)
```

Adding elements to a new dictionary from an existing dictionary.

new_dict["new key"] = old_dict.pop("old key")

dictKeys:

dictionary = {key: value}

variable = dictionary.keys()

dictValues:

dictionary = {key:value}

variable = 0

for value in dictionary.values:

variable += value

.items()

` dictionary = {key: value}

For key, value in dictionary.items():

Print("string" + str(value) + key)

Files with python:

Opening and reading files:

With : using with we can then apply the open() function to a file

.read() : the read function is used to read the contents of a file

.open(a, b) : opens a with attribute b (r – read, w – write, a – append)

e.g.

```
with open('file.extension') as file_variable:
    new_variable = file_variable.read()
    print(new_variable)
```

.readlines() : reads the file line by line

.readline() : will read the next occurring line in the file

.write() : writes what is added to the brackets into the attached file

csv.DictReader() : presents the plain text as a dictionary

json.load() : loads json file contents

json.dump(a, b) : converts a to json and saves to b

Types in Python:

type(variable) : using this function on a variable will return the type of that variable

class Class name: is used to define a class in python, class names are capitalised