Python

Handling variables

Date/Time

#Create a datetime object (from datetime import datetime, time, date)

```
mydt = datetime(int(year),int(month),int(day),int(hour),int(min),int(sec))
```

#Isolate time and date components

```
mydate = mydt.date()      mytime = mydt.time()
```

#datetime arithmetic

add a day to your mydt:(use timedelta, remember to import it)

```
mydt + timedelta(days=1)
```

Essentially: +/- timedelta(days=x, hours=x, minutes=x, seconds=x)

#Replace a date or a time:

```
mydt = mydt.replace(hour=12)
```

Print to Screen:

```
print (t)
#This is how to comment
```

Store user input:

```
var = input("please enter: ")
```

Types of Data:

Integer (23), Float (23.4), String ("hello"), Boolean (TRUE, FALSE)

Finding out what data type my variable "x" is and the length:

```
type(x)       len(x)
```

Converting to different data types:

```
str(x)   #convert  variable x into string
int(y)   #convert variable y into integer
float(z) #convert variable z into float
```

Define a variable:

```
a = 3
b = 4
t = a + b
```

String Concatenation:

combine strings and variables together:

```
x ="string1+ str(age)" + "words"
```

String Formatting

Use place holders { } in our string - to mark where variables should go to make it easier visualize output. Below are two ways to format:

```
a,b = "Bob","Ali" #One liner assignment
msg = f'We combo {a} and {b}'
msg2 ="We combo {} and {}".format(a,b)
```

Try/Except

```
try:
    <try this which could give an error>
except <the expected error>:
    <do this to show error was caught>
#you can use sys.exit() to stop the script
```

Conditions

DECISION MAKING- IF/ELSE statement:

```
If <condition>:
   <do something>

Elif <condition>:
   <do something>

Else:
   <do something>
```

DECISION MAKING- While statement:

```
While <condition>:
   < Do something>
```

DECISION MAKING- For statement:

```
For i in my_list:
   < Do something with
     respect to i>
```

Note: "i" will store the element

```
#e.g
my_list = [1,2,3]
For i in my_list:
   print(i)
```

This will print: 1 2 3

Loop through a range of values

```
for i in range(3):
   print(i)
```

This will print: 0 1 2

```
for i in range(1,4):
   print(i)
```

This will print: 1 2 3

DECISION MAKING- Boolean IF:

```
#is_hot --> boolean

If is_hot:
   #do this if true
```

Classes/functions

Functions:

Define a function:

```
Def myfunction():
    <Do something>

myfunction() # call function
```

Example:

```
def myfun(a,b):
    dump = a + b
    return dump

ans = myfun(1,2)
print("answer: {}".format(ans)
```

prints --> answer: 3

Class:

A class can have a set of variables and methods.

Inside a class, we call variables: attributes and functions: methods.

The class is a blueprint we use. In our program we create a copy, an "object" of that class. that object has access to the attributes and methods. Here is an example of a class with attributes and methods.

```
class cat:
   #Define some attributes:
   def __init__(self, name, weight):
      self.name  = name
      self.weight = weight

   #Define some methods:
   def myname(self):
      print("My name is: " + self.name)
   def myweight(self):
      print("My weight is: " + self.weight)
```

Outside the class, in the main program, create a cat object:

```
cat1 = cat("Andy","10")
#Call a method defined in the class:
cat1.myname()
```

Notes:

To create the class, we need a constructor containing the attributes followed by any functions.

To define attributes, you need this __init__(). Inside the brackets we need "self" followed by a list of all the attributes. The class will accept parameters to be used within its collection of methods

"self." Is attached to an attribute that you are referencing from the class variables. so you need to always write self.<name_of_variable>.

Data Structures

Dictionary:

We have a key-value pair in a dictionary, like so:

Key	*Value*
500	"John doe"
501	"Alice walt"

Use a dictionary to store the above information.

Define an empty dictionary:

```
dict = {}
```

Create a dictionary:

note left side are the keys which must be string or int

```
dict = {"500": "John doe", "501": "Alice walt"}
```

Look up something in the dictionary:

```
print (dict ["500"] )  # this prints:"john doe"
```

Update an entry in dictionary:

List comprehension

We can construct a new list from an old one

```
New_List = [<do this> for i in my_list]
```

Do some calculation with respect to "i", for every "i", in this predefined list.

Eg:

```
my_list = [1,2,3]
New_List = [i*2 for i in my_list]
# New_List becomes [2,4,6]
```

List:

store a collection of data (of any datatype)

```
mylist = [1,2,"hey"]  # access an element: mylist[i]
```

Add item to end:

```
mylist.append(120)
```

Add value at a certain index:

```
mylist.insert(<index>, <value>)
```

Remove or "pop" an item at a given index

```
mylist.pop(i)
```

Tuples:

Same as lists except they are immutable- cannot change

```
Mytuple = (1,2,3)
```

List unpacking:

Assign variables to elements in a list/tuple:

```
Coordinates = (1,2,3)
x,y,z = coordinates
```

Access elements in list:

```
for i in list:
    # "i" contains element
```

2D lists

```
mymatrix = [
     [1,2,3],
     [4,5,6],
     [7,8,9] ]
```

Access elements in 2D list:

```
for row in mymatrix:
   for element in row:
      print element
```

```
dict["500"] = "Harry Holland"
```