Python Beginners workshop

Day 3

A quick recap…

Functions

A function is a block of code that is run to process some input and give an output.

In programming, a certain function may have to be run multiple times throughout the program. For example, in a program where you are taking of several numbers and returning the factorial of that number, you will have to calculate the factorial multiple times. Here, you can use a function, instead of a generic block of code

A function is declared in this way:

```
def function_name(<arguments>):
    # your code
```

A function can take any number of arguments. These arguments are defined by placing variable names inside brackets next to the function name:

```
def function(arg1, arg2, …):
```

To call a function later on in the program, just do:

```
function()
```

Or, pass in arguments like this:

```
function(arg1, arg2, …)
```

A function can return values using the return keyword. Any data type can be returned by a function, and return values can either be used in conditions, or assigned to variables(example coming up)

About function arguments and returns

Not all functions need to take an argument

A function cannot have the same name as a pre-defined python function or a variable you have made

The arguments can be of any data type, and a function can take multiple arguments of different data types

Not all functions need to return a value, but if a value is returned it can be assigned to a variable, or used directly in a conditional statement

Arguments need to be passed into a function in the correct order. For example, if function f1 takes arguments as f1(x, y) then while calling it you cannot pass f1(y, x); its output will be not as expected. Look below:

```
def f1(x, y):
        # does something with x and y


# there are two variables called a and b
# call the function on a and b
f1(a, b)
```

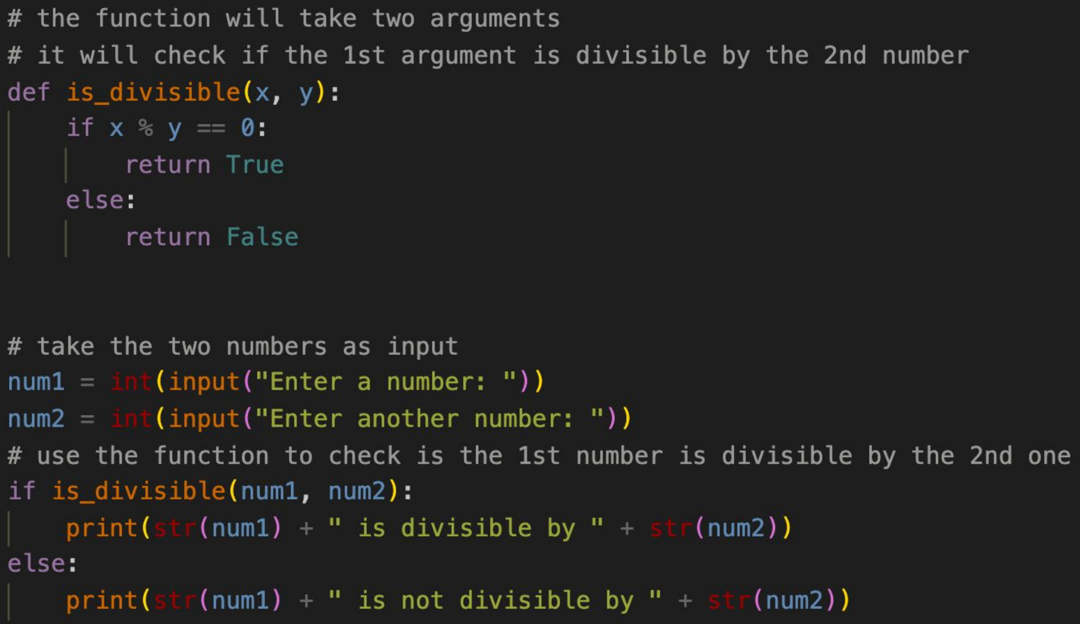
Example: Check if a number is divisible by 2 or not(even or odd)

```python
# function is called is_even, and takes one argument
def is_even(n):
    if n % 2 == 0:
        return True
    else:
        return False

# take input, and remember to typecast it from string to integer
num = int(input("Enter a number: "))
# check if the number is even(so check if function return true or false)
if is_even(num):
    print("number is even")
else:
    print("number is odd")
```

Functions with multiple arguments

Example: Input two numbers, and check if the first number is divisible by the other number



```python
# the function will take two arguments
# it will check if the 1st argument is divisible by the 2nd number
def is_divisible(x, y):
    if x % y == 0:
        return True
    else:
        return False


# take the two numbers as input
num1 = int(input("Enter a number: "))
num2 = int(input("Enter another number: "))
# use the function to check is the 1st number is divisible by the 2nd one
if is_divisible(num1, num2):
    print(str(num1) + " is divisible by " + str(num2))
else:
    print(str(num1) + " is not divisible by " + str(num2))
```

Classes

Sometimes, the data types provided to you by python may not be enough, and you may want to create your own special data types. Python allows you to do this via classes. A class can be used to store different types of information about an object in one data type, similar to a dictionary.

```
class Student:

        def __init__(self, name, age, grade, school, hobbies):

                self.name = name

                self.age = age

                self.grade = grade

                self.school = school

                self.hobbies = hobbies

me = Student(“Prakamya”, 16, “IB Yr 1”, “GIIS”, [“Tennis”, “Coding”, “Playing video games”])
```

Dot notation

This is a way of referencing the properties of a class

```
me = Student(“Prakamya”, 16, “IB Yr 1”, “GIIS”, [“Tennis”, “Coding”, “Playing video games”])
```

If I want my name

```
me.name
```

For my age:

```
me.age
```

And so on:

```
me.grade # gives my grade
me.school # gives my school
me.hobbies # gives the list of my hobbies
```

Strings, Arrays and other iterables

An iterable in python is a data type that can looped over using a for loop. They usually have indexes, which are values for the place at which an element is located

There are 5 main iterables: strings, lists, tuples, sets and dictionaries

Strings

A string is a length of any type of characters, enclosed in double or single quotes(make sure that you use either only double quotes or only single quotes throughout your program, for consistency in your code and to avoid confusions like “hello’ not being validated as a string)

Ex: “Hello”, “abc123”, “&32jii”

To access a particular character in a string, you can use indexing. This is done using square brackets []

The index is the position in the string of a character. For example, in string, the index of h is 0(indexes start with 0, so the 1st character is 0, 2nd is 1 and so on)

Indexing

string = “hello”

string[0] # will be ‘h’

string[3] # will be the 4th character, so ‘l’

string[4] # will be the 5th character, so ‘o’

string[6] # would be the 7th character, but string only has 5 characters, so this will give an IndexError: index out of range

string[5] # even though string has 5 characters, indexes start from 0, so the last character will be 5-1 = 4th index, so this will give an IndexError: string index out of range

Indexing works similarly for other iterables, using [] to reference an index(indexes always start from 0)

Negative indexes

Sometimes, you may want the last character in a string. Here, python allows for negative indexing, where an index of -1 is the last character, -2 will be the second last character, and so on:

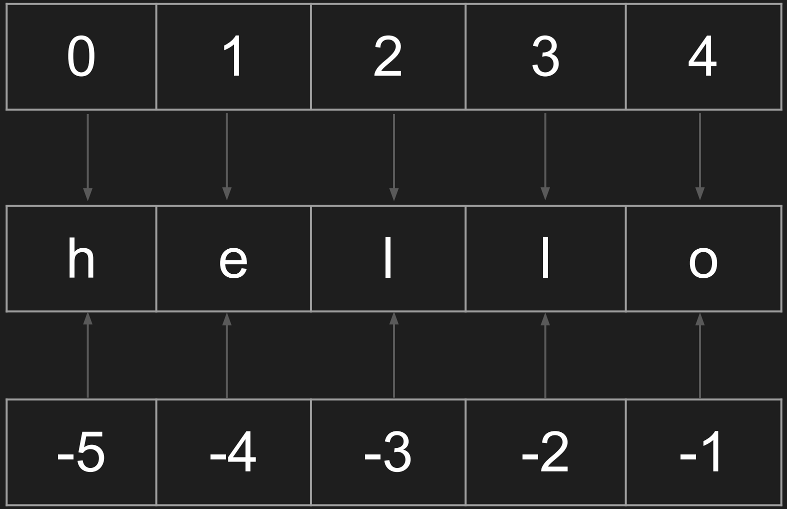
```
string = “hello”

string[-1] # will be last character, so ‘o’

string[-4] # will be 4th last character, so ‘e’

string[-9] # would be the 9th last character, but only 5 characters in total, so will give IndexError: string index out of range
```

Indexing



If the length of an array is L, then what is the index of its last element?

We can see that if an element is at the 1st position, its index is 0. So, we can deduce that if the position of an element is x, then its index is x - 1. So the last element of an array of length L will be at position L, so its index will be L - 1

Splicing

This is a method of indexing which returns a section of the string, instead of just one character

Uses the same [] notation, but the brackets will take up to 3 arguments, separated by colons (:) instead of commas

The first argument is the starting index of the section(default is 0, so leave it blank to take a splice from the beginning of the string)

The second argument is the ending index of the section(default is the last index, so leave it blank to take a splice till the end of the string)

The third argument is optional, but it takes an interval along which to splice. By default it is 1, but if the argument is provided then it will use that as the interval

Splicing also works with negative indexes

```
string = “hello world”
```

For string[3:8] index 3 is ‘l’, index 8 is ‘r’, but giving 8 as the ending argument will mean that the 8th index is not included, so it will stop at ‘o’, which is index 7

Finally, you get “lo wo”

```
string[2:11]  # index 2 to index 10, so “llo world”

string[0:10] # index 0 to index 9, so “hello worl”

string[ :10] # default for starting is 0, so index 0 to 9, so “hello worl” again

string[3: ] # default for ending value is last character, so index 3 to end, so “lo world”

string[ : ] # takes from start to end, so “hello world”. Same as getting entire string
```

string = “hello world”

For string[3:8:2], it takes from index 3 to 7 as before, but since an interval argument of 2 has been provided, it will take every 2nd index from 3, so index 3, then 5, then 7, then stop. Final output will be “l o”

string[2:10:4] # index 2 to 9, skipping 3 each time, so indexes 2 and 6 only, so “lw”

string[:10:3] # starting to index 9, skipping 2 each time, so indexes 0, 3, 6, 9, so “hlwl”

If the interval argument is negative, then it will take the splice in reverse:

For example, string[9:2:-1] will take from index 2 to 8, which is “llo wor”, but -1 will reverse it, so it will become “row oll”

string[: :-1] # will take from start to end, but -1 will reverse it, so “hello world” becomes “dlrow olleh”

Using string[: :-1] is a simple way of reversing a string

String methods

String methods are a number of functions that can be applied to strings. These functions are predefined in python, and you can call them by using dot notation:

```
name = “ajay”

# to call a string method

name.method_name()
```

In a generic form: `<variable name>.method_name(<possible arguments>)`

String methods

string.upper() makes the string uppercase

string.lower() makes the string lower case

string.capitalize() makes the first character in the string uppercase

string.isalpha() checks if all the characters in the string are alphabets

string.isdigit() checks if all the characters in the string are digits

string.isnumeric() checks if all characters in the string are numeric

string.isupper() checks if all characters in the string are uppercase

string.islower() checks if all characters in the string are lowercase

string.swapcase() converts lowercase characters into uppercase and uppercase characters into lowercase

string.index(char) returns the index of the character. If the character is not in the string, it returns an error

Python f-strings

Sometimes, you will need to output a string using print() that has a variable added to it. Previously, we did this using string concatenation and typecasting, as below:

```
print(str(x) + “ + “ str(y) + “ = “ + str(x+y))
```

Python f-strings are a way to plug in variable values into a string for easier output

Add an f at the beginning of a string, before the quotation marks. Then, write out what needs to be printed. Wherever you need to put the value of a variable, put the variable between {}

```
print(f”{x} + {y} = {x+y}”)
```

This automatically plugs the variables into the reserved “slots” in the string. The slots are defined by {}, and the f at the beginning ensures that python understands that it is a formatted string, and will substitute the variables automatically.

Lists

A list is a collection of different values that can be of any data types, separated by commas and placed within square brackets []

Lists are ordered, meaning that the order of items in a list is fixed, unless you change it manually. This also allows you to reference items in a list using indexes(which work the same way as they do in strings)

They are also mutable, which means that values in a list can be changed any time, and you can add or remove elements from a list as well

Ex: [“help”, 193, 9.0, False]

Indexing and splicing works on lists the same way it does on strings

Finding the length of an iterable

Often in many programs, you will require the length of an array, maybe for error-checking of some kind(4 values need to be entered into a list, so check the list to see if 4 values are there) or maybe to loop through the iterable

Python provides the len() function that takes an iterable, and returns its length(the number of elements in the iterable)

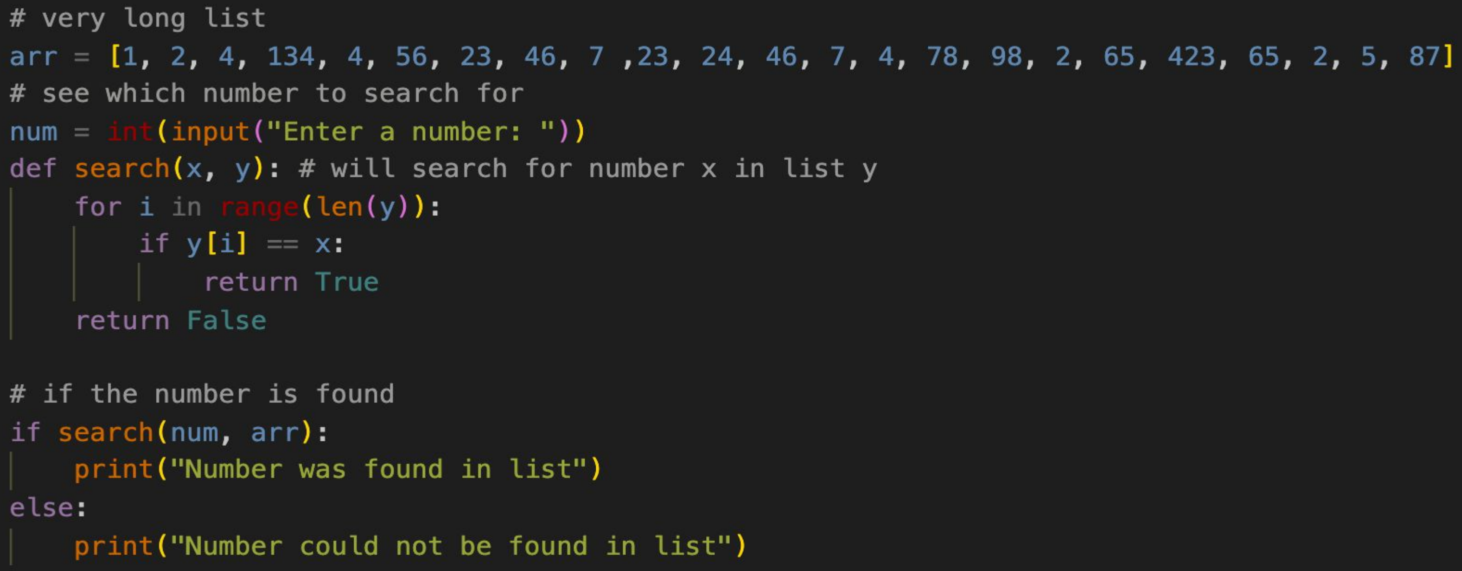
It can be used like this:

```
print(len(x)) # where x is a string, set, list or tuple
```

For a string like “hello world” its length is the number of characters in the string(in this case 11) and for a list like [2, 3, 4, “hello”, True] its length is the number of comma-separated values(in this case 5; same for a tuple or set)

Using the len() function

Let’s say you need to write a program to search for a number in a list of numbers. The simplest way to do this is using a for loop, to iterate over each element, until you find the element. For this, you need to know the length of the list.



```python
# very long list
arr = [1, 2, 4, 134, 4, 56, 23, 46, 7 ,23, 24, 46, 7, 4, 78, 98, 2, 65, 423, 65, 2, 5, 87]
# see which number to search for
num = int(input("Enter a number: "))
def search(x, y): # will search for number x in list y
    for i in range(len(y)):
        if y[i] == x:
            return True
    return False

# if the number is found
if search(num, arr):
    print("Number was found in list")
else:
    print("Number could not be found in list")
```

For loops and iterables

We have seen that you can use the statement `for i in range(len(arr))` to loop through an iterable(string/tuple/set/list), but python provides a simple shorthand for this as well.

Instead of having to loop through a range of values using `range()`, you can just loop through the variable directly:

```
arr = [2, 5, 3, 6, 67, 5, 2]
for i in arr:
        print(i) # this will print 2, then 5,  then 3, and so on until the end of the list
```

This way, the length of the variable is not required

In keyword

This is a logical operator to check if a value is present in an iterable, and returns True if it is, and False if it isn’t

```
arr = [2, 54, 23, 456, 32, 78, 8]
if 10 in arr:
        print(“10 is there in the array”)
else:
        print(“10 is not there in the array”)
```

List methods

These are used in the same way as string method; using dot notation

arr.append(value) # adds the value to the end of the list

arr.clear() # removes all elements from the list

arr.count(value) # returns the number of times value is repeated in the list

arr.index(value) # returns the index of the given value; -1 if the value is not present

arr.insert(index, value) # inserts value at the given index

arr.pop(index) # removes the element at the given index

arr.remove(value) # removes the given value from the list(if it is repeated, it removes the first one it finds)

arr.reverse() # reverses the list

arr.sort() # sorts the list in ascending order. To sort in descending order, use arr.sort(reverse=True)

Dictionaries

A dictionary in python is a set of key-value pairs. A key is like a custom index, except that it represents a key, instead of the position of an element. A key can be mapped to a value using a colon (:) and each key value pair is separated by a comma.

```
me = {“name”: “Prakamya”, “age”: 16, “school”: “GIIS”}

me[“name”]  # “Prakamya”

me[“age”]  # 16

me[“school”]  # “GIIS”
```

Iterating over dictionaries

The len() function does not work on dictionaries, and for loops will not work either, so to iterate over a dictionary, you must use its keys or values. These can be accessed using dot notation

```
me = {“name”: “Prakamya”, “age”: 16, “school”: “GIIS”}

for i in me: # will not work

for key in me.keys(): # creates an array with the keys of the dictionary, and iterates over it
        print(key) # will print out name, then age, then school
        print(me[key]) # will print out Prakamya, 16, GIIS

for value in me.values(): # creates an array with the values, and iterates over it
        print(value) # will print out Prakamya, 16, GIIS
```

Changing and getting dictionary values

To add values to a dictionary, you can just create a new key-value pair:

```
me = {“name”: “Prakamya”, “age”: 16, “school”: “GIIS”}
me[“class”] = “IB Yr 1” # “class”: “IB Yr 1” gets added to dictionary
```

To remove an item from a dictionary, use the .pop() method:

```
me.pop(“class”) # removes the “class”: “IB Yr 1” key-value pair
```

There are two ways to get an item from a dictionary. One is to treat keys as indexes:

```
me[“name”]  # “Prakamya”
```

The other is to use the .get() method:

```
me.get(“name”)  # “Prakamya”
```

Python modules

Modules are files that contain code that may be useful for you. They contain functions that are not in-built in python, and have to be imported from elsewhere. Some modules are pre-installed with python, but others need to be installed using pip - python package installer.

Some modules included in python:

Time, Math, Random, OS

Some modules that need to be pip installed:

MPU, Pandas, NumPy

To use a module in your python program, use the import keyword:

```
import <module_name>
```

Modules often have useful functions inside them, so to use any of these functions, you can use dot notation:

```
<module_name>.<function_name>
```

Some modules are very big, and have many functions, but you may want to use only a few of those functions. In this case, you can use the from keyword to choose which function to import:

```
from <module_name> import <function_name>

<function_name>  # you can use the function without dot notation
```

Time module

```python
import time

time.time() # gives the number of seconds since Jan 1st, 1970 00:00:00(start of Unix epoch)

time.ctime(seconds) # takes the number of seconds since the start of Unix epoch as argument, and returns the formatted time at that number of seconds

time.localtime() # gives the local time as a time_struct data type

time.strftime(format, time_struct) # gives the local time(passed in as time_struct, usually from time.localtime()) in a customizable string format

time.sleep(seconds) # pauses the program for some seconds
```

Math module

```
import math

math.sqrt(num) # returns the square root of a number

math.pi  # returns the value of pi to 15 decimal places

math.gcd(a, b) # returns the HCF of a and b, given that they are integers

math.lcm(a, b) # returns the LCM of a and b, given that they are integers

math.ceil(num) # rounds a decimal number up to the next integer

math.floor(num) # round a decimal number down to the previous integer

math.factorial(num) # returns the factorial of a number
```

Random module

```
import random

random.random() # returns a random decimal between 0 and 1

random.randrange(a, b) # returns a random decimal between a and b

random.randint(a, b) # returns a random integer between a and b

random.choice(iter) # takes a tuple/set/list and returns a random value from it

random.choices(iter) # takes a tuple/set/list and returns a selection of random values
from it

random.shuffle(iter) # takes a tuple/set/list, shuffles it and returns the shuffled variable
```

Colorama module

This module needs to be installed via pip: pip install colorama

```
from colorama import Fore, Style, Back, init
init(autoreset=True)

print(f"{Fore.RED}Hello")   # prints red text

print(f"{Back.GREEN}Hello")   # prints text with green background

print(f"{Style.BRIGHT}Hello")   # prints brighter text
```

You can look at the documentation(link in resources slide) to see what colours are available for use

The 6 subject marks of 30 students from a class need to be input into a program, and the average of each student needs to be calculated, along with their overall grade. Write a program to output the name of the student, their marks, their average and their grade. Store all these values in a data structure of your choice. Use the grade boundaries below.

Average of student	**Grade assigned**
90 and above	A
80 to 90	B
70 to 80	C
60 to 70	D
50 to 60	E
49 and below	F

Resources

Everything Python

Python functions and arguments

Python classes

Python dot notation

Python strings

Python f-strings

Python indexing and splicing

Python lists

Python len() function

For loops with iterables

Python dictionaries

Python dictionary manipulation

Python modules intro

Python time module

Python math module

Python random module

Python colorama intro