python

Python Beginners workshop

Day 2

A quick recap…

Our first program - input, output and variables

This program features some simple output:

```
print(“hello world”)
```

Now, let’s make it say hello to you:

```
print(“hello ajay”)
```

But what if your name is not Ajay? You need to take input from the user:

```
name = input(“What is your name?”)
```

But what to do with that name? You need to print it out, along with a hello:

```
print(“hello “ + name)
```

Data types in Python

Strings

Any length of any type of characters: alphabets, numbers, punctuation etc.

Always put them in quotation marks(single or double)

Ex: “Hello”, “abc123”, “for(3/]fwe[}e3q”

Numbers(integers and floats)

Integers are whole numbers such as 10, -231, 329, 0

Floats are decimal numbers such as 56.23, -1.23, 0.222

Boolean

Can only have 2 values: True or False

List

The python version of an array; you can put a combination of any number of values of any data types into a python list.

Just separate the values using commas, and put them inside square brackets []. Can be empty, and is referred to as a null list

Ex: [10, “hello”, 3.4, False, “ab90”]

Sets

Similar to a List, but the order of items changes constantly, so not easy to manipulate or access values in a set

Tuple:

Similar to a List, but you cannot change the values once they have been set

To make a tuple, use () instead of [] and separate values using commas

Ex: (1, 90, “hi”, True)

Dictionary:

This is a way to map key-value pairs in python

You can assign a value to a key; the value can be of any data type, but the key has to be a string

Usually used to store multiple types of information about an object; uses {}

Ex:

```
me = {“age”: 16, “class”: “IB Yr 1”,
“school”: “GIIS”}
```

Typecasting

Let’s say you have a variable x with a value of “19”. This is a string, with the number 19 inside it. However, you want to calculate something using that number. Since you cannot perform arithmetic operations on strings, you have to change its data type from string to integer. This is called typecasting, and is done this way:

```
x = “19”

y = int(x) # returns the string in an integer format, so y = 19
```

To check what data type a variable’s value is, you can use the type() function:

```
print(type(x)) # output will be <class ‘str’>, indicating that x is a string

print(type(y)) # output will be <class ‘int’>, indicating that y is an integer
```

Similar typecasting can be done to other data types in python, provided that converting will result in a valid value for the data type:

```
x = “1.3”

y = float(x) # int() will not work, because 1.3 is not an integer

a = 45

b = str(45) # converts 45 to “45”

p = “True”

q = bool(p) # converts string “True” to boolean True
```

There is also tuple(), list(), set() and dict() to convert to tuple, list, set or dictionary respectively

Operations on variables

There are 3 types of operations: arithmetic, comparison and logical operations

Arithmetic operations: +, -, *, / etc.

Comparison operations: <, >, == etc.

Logical operations: and, or, not

You can use these operations like you use them in real life

All computers follow the BODMAS rule: brackets first, then exponents, then division, then multiplication, then addition, then subtraction

You can only use () for calculations, [] and {} are for List and Dictionary respectively

Arithmetic operations

Addition (+)

Ex: a = 10 + 32

Answer: a = 42

Subtraction (-)

Ex: b = 434 - 12

Answer: b = 422

Multiplication (*)

Ex: x = 10 * 43

Answer: x = 430

Division (/)

Ex: y = 10/3

Answer: y = 3.33333

Additional Python arithmetic

Exponents (**)

Ex: a = 4**3

4**3 means 4 * 4 * 4

Answer: a = 64

Floor division (//)

Returns the whole number part of the division, ignoring the remainder

Ex: b = 10//3

Answer: b = 3

Modulus (%)

Returns the remainder of the division

Ex: x = 10%3

Answer: x = 1

String/List concatenation

You can add two strings/list to each other, which will combine them

[1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]

“Hello” + “world” = “Helloworld”

Comparison operations

Greater/Lesser than

Uses the arrow brackets: <, >

Ex: 10 > 5 is True

6 < 3 is False

Adding an = sign will change the operator to also check if the values are equal:

5 <= 10 is True

5 >= 5 is also True, since 5 = 5

Exactly equal to

Normally, you use an = sign, but in python you use ==

Ex: 4 == 4 is True

“Hello” == “hello” is False (we consider capital and small letters to be different)

Not equal to

You can add an ! to an = sign, to denote != (not equal to)

Ex: 4 != 9 is True

“Hello” != “Hello” is False

Logical Operators

and

Checks if all conditions provided to it are True

Ex: 5 > 4 and 9 < 10 is True

5 < 4 and 9 > 10 is False

5 < 4 and 9 < 10 is False, since both conditions need to be True

not

Returns the reverse of the condition

Ex: 5 > 4 is True, but not 5 > 4 is False

or

Checks if any one of the conditions provided to it are True

Ex: 5 > 4 or 9 < 10 is True

5 < 4 or 9 > 10 is False

5 < 4 or 9 < 10 is True, since only one of the conditions need to be True

Conditions and Conditionals

A conditional is a statement that checks if a given condition is true or not

If the condition is true, a certain functions is run; else a different function is run

In python, you can check conditions this way:

```
if <condition>:
    # do something
else:
    # do something else
```

You can use any operators to make your conditions, and you can check for multiple conditions

For example, this is some code to check if a number is positive or negative. If it is positive, then it checks if the number is divisible by 2(odd or even)

```python
# if a number is divisible by 2, the number / 2 will have remainder 0
# % is modulus operator, which gives the remainder of the division
if number < 0:
    print("number is negative")
elif number % 2 == 0:
    print("number is even")
else:
    print("number is odd")
```

Loops

Sometimes, you may need to repeat something a number of times. For this, you have 2 types of loops in python: for and while loops

A for loop will repeat something a specific number of times

A while loop will repeat something forever, while a condition(that is provided to it) is satisfied

```
for <counter variable> in <range>:

        # do something

while <condition>:

        # do something
```

For loops

In a for loop, you need to provide a counter variable, and a range of values that it needs to go through

Use the range function for this: `range(0, 10)` means that the counter variable will go through 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and stop

`range()` takes 3 arguments: starting value(default is 0), ending value, and step(default is 1)

`range(5)` will go through 0, 1, 2, 3, 4

`range(3, 6)` will go through 3, 4, 5

`range(2, 15, 3)` will go through 2, 5, 8, 11, 14 (adds 3 instead of 1)

The counter variable can be used to keep track of where you are in the loop, and doesn't need to be declared beforehand

```
for i in range(10):
        print(i)  # will print 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

While loops

In a while loop, you have a function repeating while a condition is true. The condition can consist of any valid combination of operators

Make sure to give a condition that will be true up to a point; otherwise, an infinite loop occurs, which can break your program, and possibly freeze up your system(worst case scenario)

Infinite loop example:

```
while 3 > 2:
        print(“hello”)
```

Example question

Find the sum of a series of numbers that are input. If the number 0 is input, stop the program

```python
# you need to input the first number
answer = int(input(": "))
# initiate the variable that will be used to calculate the total
total = 0
# if 0 is input, then the loop is stopped
# so stay in the loop while answer is not equal to 0
# remember that != is not equal to operator
while answer != 0:
    total = total + answer
    answer = int(input(": "))
print(total)
```

Shortcut assignments

If you are performing arithmetic operations on a variable, then shorthand assignments can be used:

x = x + y becomes x += y

x = x - y becomes x -= y

x = x * y becomes x *= y

x = x/y becomes x /= y

x = x//y becomes x //= y

x = x%y becomes x %= y

x = x**y becomes x **= y

A student’s marks need to be entered and his average calculated and output. 6 subject marks will be input, each out of 100. Using the grade boundaries below, write a program to output the average and the grade of the student

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 data types

Python typecasting

Python operators and operations

Python conditionals

Python for loops

Python while loops