**Input and Output**

***Input*** (noun): data that is entered into the CPU

***Input*** (verb): the act of entering data into the CPU

You can input data from many different input devices:

* the keyboard
* the mouse
* a game controller
* a microphone
* a video camera
* a hard drive/USB stick
* an internet connection

***Output*** (noun): data that is sent out from the CPU

***Output*** (verb): the act of sending information out from the CPU

The CPU can send output to many different output devices:

* A monitor
* a game controller (vibrations)
* a speaker
* a hard drive/ USB stick
* an internet connection

**Output to the Monitor**

To write words on the screen, we use the ***print()*** function:

print ("hello")

**Input from the Keyboard**

We can get input from the keyboard using the input function:

input()

The input function gets the computer to wait for the user to enter a value from the keyboard. If you run the code, nothing special will appear to happen. A cursor will blink in the output window as the code waits patiently for you to enter something. You can type anything, and then hit the ENTER key. When you do this the program will end, indicated by three arrows in the left margin.

Try this:

input()

print ("hello")

Now run it. You’ll notice that the second line (print (“hello”)) won’t execute until you hit the enter key. It doesn’t matter what you type as long as you hit the enter key.

Let’s add instructions inside the parentheses of our input function. Try this:

input("please enter your name: ")

print("Thank you!")

Now the code gives instructions the user can understand. Run it and see that it waits patiently for you to enter a name. Notice that you don’t have to actually print a name, since the computer does not know what a name is:

>>> Please enter your name: 3.1415

>>> Thank you!

The real power of the input function is when we store your response in a variable. Here’s how:

x = input() # get input from keyboard

print (x) # print the input

This version stores the user’s input in the variable called x. The second line prints the contents of x.

**Data Types**

The computer recognizes different data. Three *primitive* data types are:

**string data**: a string of characters. Example: x = “banana1232”

**integer data**: positive and negative whole numbers. Example: x = 23

**floating point data**: any number with a decimal. Example: x = 3.56

See what this does:

age = input("please enter your age: ") # input

print ("You are", age, "years old!") # output

Try the above code out. Try entering a word instead of an age:

please enter your age: infinite

You are infinite years old!

Obviously, this is not the way we meant the code to work. The input() function considers the user input to be a ***string*** by default, which means we can type anything, including letters, numbers, punctuation. If you want to ensure the user is entering a number, we must specify the data type:

age = int (input("please enter your age: ")) # must be an int

shoeSize = float(input("please enter your shoe size: ")) # must be a float

name = str (input("please enter your name: ")) # must be a string

print ("hello, ", name, "!")

print ("you are", age, "years old!")

print ("and your shoe size is", shoeSize)

Now if you test the above code by giving nonsensical input, it will trigger an error.

**Comments**

When our programs get more developed, we will want to put ***comments*** in them to explain the tricky parts. Use the pound sign (#) to begin a comment:

myAge = 18 # remember my age

print (myAge) # print it out!

***Comments*** are statements written to make the code more understandable. They are ignored by the interpreter and are not converted to binary machine code.

**Formatting Output**

Sometimes you want to format your output in a certain way. For example:

a = 40

b = 3

c = a/b # this divides two integers

print (c)

Now compare to this:

a = 40

b = 3

c = a//b # this rounds down (called floor division)

print (c)

Finally, we can round off in the print statement:

a = 40.0

b = 3.0

c = a/b

print ("%.2f" % c) # this cuts off after two decimal places

This is a little confusing, but basically says to print a number with 2 digits after the decimal, using the variable c.

**String Concatenation**

Strings can be added:

String addition: “4” + “5” + “6” = “456”

Integers add differently than strings:

Integer addition: 4 + 5 + 6 = 15

If you don’t specify what type of data you are using, you might get a confusing result.

a = input("please enter a number: ")

b = input("please enter a number: ")

print ( "The sum is: ", a + b)

This isn’t going to do what you think it should. Can you see why?

**A Note on Quotation Marks**

Python doesn’t distinguish between the two types of quotation marks. You can use single or double quotes for anything:

print ('hello') # ok - single quotes

print ("hello") # ok - double quotes

# print ('hello") # not ok - don’t mix quotes

# print ("this is a "quote" in quotes!") # not ok - can't put quotes inside quotes

print ('this is a "quote" in quotes!') # ok - use double quotes inside single quotes

Python is different from most languages in this regard. If you get used to switching your quotation marks around, you are going to have trouble when you move to another language. For the remainder of these lessons, we will stick to double quotes for most things.

**The New Line Character**

Try this:

print("My name \n is Sam")

When executed, the output looks like this:

My name

is Sam

There is a special character code (called an escape character) inserted into the string. This particular escape character adds a new line in the middle of the string. You can also get the same effect by printing each line separately:

print("My name")

print ("is Sam")

Here are other escape characters, for example:

print("My name \t is Sam")

The above example adds a tab (usually 4 spaces in Python) to the printout.

**Printing on a Single Line**

Python assumes that the programmer wants two separate print statements to print on two separate lines, like the above example. Sometimes you want to connect two separate print statements into a single line of output. To do this, you have to tell the print function not to end the string with a new line. Here’s how:

print("My name", end = "")

print ("is Sam")

The print function has a special parameter (end = "" )you can insert into the print statement that states how you want the output to end. The default is to end with a new line. In the above case, we have asked to end with nothing (the quotes are empty). The result is this:

My name is Sam

You can end a print statement with anything. Try this:

print("My name", end = "zzzzzzzzzzzz")

print ("is Sam", end = " !!!!!!!!!!!")

**Multiple prints**

What does this do?

print("Hello", "goodbye. ","P.S.","I’m shy")

The print() function allows you to print multiple things along one line. Each item must be separated by a comma (shown below in blue):

x = 23

y = "green"

print("Hello my age is ", x, "and my favourite colour is", y)

Notice the variables (x and y) are not inside the quotes. See the difference?

x = 23

print("Hello my age is, x")

**Exercises**

1. Predict the output of the following code. Paste to the python interpreter and run to see if you are correct.

print (a)

a = 5

Explain what happens and why.

1. Predict and explain:

name = "Mariam"

print ("My name is ", end = "")

print (name)

If you change the second line to this, how will it be different?

print ("My name is ")

1. Write a short program that asks the user for two numbers (separately) and then prints out their sum.

Enter a number: <23>

Enter a second number: <12>

The sum of 23 and 12 is 35.

1. Write code that will ask the user for the distance traveled and the speed traveled, and then calculate the time of travel.

Enter the distance you are traveling in km: <420>

Enter a speed of travel in km/h: <90>

The trip will take 4.6 hours

Note: Round to one decimal place. See the example in the lesson.

**Keywords: *Input, output, print, case sensitive, whitespaces, variable, literal, string, integer, floating point, assignment statement, comments***