Week 4

Input

Input from terminal can be received using `input()` function in python. `input()` accepts an argument which may be a string, int or float. Then performs the following operations:

- Writes the argument to standard output
- Reads from the standard input of the terminal until the next line character is reached
- Returns the received input as a string And that's all with `input()`.

Writing Returned string from `input()` to `stdout`

```
input("Enter")
```

The above line of code will successfully read input from `stdin` and `return` it. The next task is to display the string to the user or more precisely write the string to the standard output stream (`stdout`).

```
print(input("Enter"))
```

> Enjoyable Task: Is it possible to write the returned string from `input()` function to `stdout` without using `print()` function?

More than one inputs can be received as follows

```
print(input("Enter n1: "), input("Enter n2: "))
```

Language of Standard Input Stream and Standard Output Stream

`print()` and `input()` function kind of automate what is done in `c++` using `cout` and `cin` statements. Let's study which of `cout`, `cin` statements are invoked when `print()` and `input()` function are called.

```
input("Enter")
```

The flow of the above statement is as follows

Function	In Program Action	Terminal
input()	Writes	stdout
input()	Reads	stdin
input()	Returns	

Similarly for a more complex statement

```
print(input("Enter"))
```

Function	In Program Action	Terminal
input()	Writes	stdout
input()	Reads	stdin
input()	Returns	
print()	Writes	stdout

```
print(int(input("Enter n1: ")) + input("Enter n2: "))
```

Function	In Program Action	Terminal
input()	Writes	stdout
input()	Reads	stdin
input()	Returns	
int()	Returns	
input()	Writes	stdout
input()	Reads	stdin
input()	Returns	
output()	Writes	stdout

String Concatenation

Concatenation is just a fancy word for combining or adding up two strings to make a single string. Therefore

```
>>> "a" + "b"
'ab'
```

Combining the strings `"a"` and `"b"` makes them `"ab"`. This operation is called string concatenation.

```
>>> print(input("Enter n1:") + input("Enter n2:"))
Enter n1:2
Enter n2:3
23
```

Although the expected answer was 5, the answer received is 23. This happened because `input()` always returns string. Therefore, + operator operated on two strings performs string concatenation and 23 is written to `stdout` as a string. So how do integer addition can be perform on data returned by input()?

Type Conversion

Strings can be converted to integers (but not all!) using `int()` function. `int()` function accepts an arugment (a string) and returns an integer. Therefore, in order to perform integer addition on data received from `input()`

```
>>> print(int(input("Enter n1:")) + int(input("Enter n2:")))
Enter n1:2
Enter n2:3
5
```

Now that there are two integers on both sides of the `+` operator, integer addition is performed.

Variables

Any defined integer, float, string, function etc all are saved in computer main memory (RAM). As explained in earlier lectures, an application level process requests the Operating System to assign some memory to store it's values.

```
5
```

The above written statement is a fully function python script. Once this process is executed, it needs to store the integer `5` in computer's memory. The Operating System will assign the process that memory somewhere in the RAM, and the process will write `5` in that memory block. Now it is practically impossible to retrieve that `5` out of memory. Computer's Memory is so large, thousands of `5` maybe saved by different processes in the

memory. Therefore, we can never exactly know where did the process saved that integer, so that we can reuse it. Variables solve this issue. Variables keep track of memory where the process saved the integer.

```
x = 5
```

In the above python statement, `x` will point to the memory location where `5` is saved.

Control Flow

Computer Programs have sequential flow. But there are situations where this behaviour needs to be changed. Flow of program can be controlled using `if ... else ...` and loops `for`, `while`.

if Statements in Python

`if` block controls whether or not to execute one or more statement based on a condition.

```
if 1 == 1:
    print("1 is indeed equal to 1")
```

`==` comparison operator compare the objects and if they are same returns True, otherwise False. So we can see very clearly that `if` sees the True or False. It is not `if`'s responsibility to compare the two objects. `==` compares the two objects and return if they are same or not. Therefore, we can also write:

```
a = 1 == 1
if a:
    print("1 is indeed equal to 1")
```

`1==1` returns boolean value which is `True`, `True` is stored in computer's RAM and `a` keeps track of it in the memory. `if` sees there's a `True`, so the program executes the indented part of code. `indented`?

Indentation

Indentation serves the same purpose which `{}` served in `C++`. That is, to specify these statements come under the hood of `if`. They are meant to be executed if and only if `if` receives a `True`. Indentation means 4 spaces before the python statement.

```
a = 1 == 1
if a:
    print("1 is indeed equal to 1")
    print("These statements are inside of if block")
print("This is outside of if block")
```

As soon as a statement starts from the very starting of a line, it is taken as a sequential python statement i.e. will be executed regardless of `if`.

> Enjoyable Task: Can tab be used to input 4 spaces in order to indent python statement? If pressing tab same as pressing 4 spaces?

if else

Indented statements are executed in case of condition in preceding `if` is `True`. In case of condition not being `True`, `else` block is executed if present (else is not compulsory with if)

```
if False:
    print("This statement will not execute")
else:
    print("This statement will execute")
```

There is another block written with else if (`elif`) in case of more than one conditions needs to be checked.

```
if False:
    print("This statement will not execute")
elif False:
    print("This statement will not execute")
elif True:
    print("This statement will execute")
else:
    print("This statement will execute")
```

In the above python script, the 2nd `elif` block will execute and no other.

Loops

Loops are used to run the same statement specified number of times. That number often depends upon some condition. `for` and `while` loops are used for this purpose.

for loop

`for` loops executes python statements specified number of times.

```
for i in range(5):
    ...
```

- for → initiates the loop
- i → is the iterator variable similar to for(int i=0; i<5; i++) in c++
- range(5) → produces an iterable: 0, 1, 2, 3, 4
- in → tells Python "take each element from this iterable and assign it to i" during each loop cycle

`range()` function will be more studied in future. For now all that is meant to be understood is range generates a series of numbers and i iterates over them.

range(start, stop, step)

Well the future is now. `range()` function takes 3 arguments, `start`, `stop` and `step`. `start` specifies where to start from including the number itself, `stop` specifies where to stop excluding the number itself and `step` specifies difference. All the arguments must be integers in range() function. Range() does not support floats.

> Enjoyable Task: Functions only return a single value, then how does `range()` return a number of values? (Note: No data structure like arrays is being used).

Back to for loop

```
for i in range(0,5):
    print(i)
```

The above written `for` loops make the statement `print(i)` run 5 times with each time i being `0,1,2,3,4`. Therefore, will generate the following result

```
0
1
2
3
4
```

while loop

`while` loops executes python statements until a condition is `True`. As soon as the condition is `False`, `while` loops stops.

```
while True:
    print(1)
```

> Enjoyable Task: The above code generates a never ending process. How can that process be stopped without closing the terminal/cmd?

Practice

> What does the input() function in Python do? Describe its three main operations.

Allowed Objects: input() and print()

> Write single python statement that:
>
> - Asks the user for their name.

> - Write the string `"Hello, userentered_name"` to standard ouput stream

> Write single python statement that:
>
> - Reads two strings from stdin
> - Writes the two strings on a single line (i.e. includes no next line character) separated by space and ends with dot `"."` to stdout.

> Write single python statement that:
>
> - Reads 4 strings from stdin
> - Adds/Combines/Concatenates strings together such that there is a space between them but the result is a single string. (Hint : Add " " single space after each input)

> Write a single Python statement that reads a city and country then prints "City: X, Country: Y". Where X,Y are read from stdin.

Allowed Objects: print(), input(), and int()

> Write single python statement that solves the Math Expression `n*a+2*b+7/c` where n,a,b,c are read from stdin.

> Write single python statement that solves the expression `(3*a+7*b)/(8*c + 8)`, where a,b,c are read from stdin, and writes the result as `"Result = calculated_answer"`
>
> - Special Test Case: c = -1

> Write single python statement that:
>
> - Solves the Math Expression `(2**a + b**c)/d` where a,b,c,d are read from stdin
> - Writes the returned int/floating value to stdout

Allowed Objects: print(), input(), int(), `if elif else` block

> Write python script that: (Hint : Read isinstance() function online)
>
> - Reads from stdin
> - writes `"This is a string"` if what is read from stdin is indeed a string (check datatype)

> Write python script that: (Hint : Read isinstance() function online) (Strict Rule: No int() allowed)
>
> - Reads from stdin
> - - Writes `"This is an integer"` if what is read from stdin is an integer (check datatype)
> - - Writes `"This is not an integer"` if what is read from stdin is not an integer

> Write python script that:
>
> - Reads a single integer from stdin
> - - sends the string "Number is even" if read integer is even
> - - sends the string "Number is odd" if read integer is odd

> Write python script that:
>
> - Compares the x and y i.e. x == y, where x and y are read from stdin.

- Writes the string `"They are same"` in case x and y are same
- Writes the string `"They are different"` in case x and y are not equal

Write a python script that: (Strict rule: No variable should be used)

- Solves two Math expressions `(a**2 - b**2)/(c - d)` and `e+f`, where a,b,c,d,e,f are read from stdin
- Compare the results of the expressions.
- Writes to stdout:
- `"Math Expressions are equal"` in case of True
- `"Math Expresssions are not equal"` in case of False

Special test Case : a = c and b = d

Allowed Objects: print(), input(), int(), while loop

Write python script that: (Strict rule: No variable should be used) (Hint : 0 integer is considered `False` while all other numbers are considered `True`)

- Keeps Reading from stdin
- As long as user enters anything other than `0`, script sends the string `"I executed"` to stdout
- As soon as user enters `0`, script stops it's execution

Write python script that: (Hint : user `break` statement to end the loop)

- Reads from stdin
- As long as user enters random strings, script keeps running (i.e. keeps reading from stdin)
- As soon as user enter the string `"stop"`, script stops it's execution

Allowed Objects: print(), input(), int(), for loop

Write python script that:

- sends the numbers 10 to 20 to stdout with step size of 2

Write python script that:

- for loop iterates through numbers 1 to 100 with step size of 1
- sends only prime numbers to stdout
  condition: Numbers must be comma separated and no next line character at the end

Write python script that:

- Iterates through numbers from 50 to 100 with step size of 10
- At each reads string from stdin
- Writes to stdout `"user entered string" iteration number`

Write python script that:

- Reads `start`, `stop`, `step` values of range function from stdin
- Writes iteration values to stdout

Allowed Objects: print(), input(), int(), while, for loop

Write python script that:

> - Reads number from stdin
> - Writes `"1,3,,5,7,9"` entered number of times to stdout
> - Note: Only a single integer should be sent to stdout using `print()`