Hi guys, thanks for stciking with me this long. My name is Akshansh(+91 8384891269, akshanshofficial@gmail.com (mailto:akshanshofficial@gmail.com)) and we are going to hop in to our new topic which is red hot. Our new topic is Functions in python. Let's find out what it is and how it works.

Why functions are needed:

1)Creating a new function gives you an opportunity ti name a group of statements which makes your program easy to read and debug. 2) Functions can make prigram smaller by eliminating the repetitive code. If you have to make a change, you just need to make it once inside the function. 3)Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into working whole. 4)Well designed functions are often useful for many programs. Once you write and debug one, you can reuse it.

Python has a math module that provides most of the familiar mathematical functions. A module is a file that contains a collection of the related fucntions.First we have to import math module so that we can use it's functions.

In [1]:

```
import math
```

In [2]:

```
math

```

Out[2]:

```
<module 'math' (built-in)>
```

you can see that math is a module and it is built in

the module objects contains the fnc and variables defined in the module. To access one of the functions, you have to specify the name of the module and the name of the function, separated by dot(also known as a period). This is called dot notation.

In [4]:

```
degrees=45
radians=degrees/180.0*math.pi
math.sin(radians)

```

Out[4]:

```
0.7071067811865475
```

in this example we used pi functions from math module and for that we had used dot notation. First we took value in degrees and then convert it into radians. You must have remember sin from trigonometry. We used radians and converted it into sin.

composition

so far, we have looked at the elements of a program - variables,expressions, and statements- in isolation without talking about how to combine them.

One of the most useful features of programming languages is their ability to take small building blocks and compose them. In example below, look and observe how I've used math module and fetched sin and pi from it and also used * to multiply and / to divide. In given example I've compose them into one line. That's the beauty of the coding (and sonetime headache too he he he)

In [5]:

```
x=math.sin(degrees/360.0*2*math.pi)
```

adding new functions

A functions definition specifies the name of the function and the sequence of the statements that run when the function is being called.

In [6]:

```
def print_lyrics():
    print("I am in so love with her.")
    print("She had me in a moment and took my heart. <3")
    print("ALthough I don't know her, but she is the most beutiful girl.")
```

def is a func keyword that indicates that this is a func definition. The name of the function is print_lyrics. The empty parentheses after the name indicates that func doesn't take any argument.

The first line of the function definition is called the header; rest is called the body. Header has to end with a colon and the body has to be indented. By convention indentation in python is always 4 spaces or you just can press fucking Tab in your keyborad.

definition of a function creates a function object, which has type function:

In [7]:

```
print(print_lyrics)
```

```
<function print_lyrics at 0x7f76f0d8d0d0>
```

In [8]:

```
type(print_lyrics)
```

Out[8]:

```
function
```

In [9]:

```
print_lyrics()
```

```
I am in so love with her.
She had me in a moment and took my heart. <3
ALthough I don't know her, but she is the most beutiful girl.
```

you can reapeat the lyrics and make it chorous by repeating the "call" of function

In [10]:

```
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
```

In [11]:

```
repeat_lyrics()
```

```
I am in so love with her.
She had me in a moment and took my heart. <3
ALthough I don't know her, but she is the most beutiful girl.
I am in so love with her.
She had me in a moment and took my heart. <3
ALthough I don't know her, but she is the most beutiful girl.
```

But this is not the way they write songs, right he he he. We are programmers not lyricist, anyways!

To ensure that a function is defined before its first use, you have to know the order statements run in, which is called the flow of execution.

Parameters and Arguments

Some of the functions we have seen require arguments. For example, when you call math.sin, you pass a number as an argument. Some functions take more than one arguments: math.pow takes two, the base and the exponent.

Inside the function, the argument areassigned to the variables called parameters. Here is a definition for a function that takes an argument:

In [18]:

```
def print_twice(argument):
    print(argument)
    print(argument)
```

Function written above takes an argument and then it print same argument twice with the help of two print statements. Let me pass one argument "Mia Khalifa" in to it.

In [21]:

```
print_twice("Mia Khalifa")
```

```
Mia Khalifa
Mia Khalifa
```

you can pass an integer too or a float whatever you want to print.

In [22]:

```
print_twice(45)
```

```
45
45
```

In [23]:

```
print_twice(39.8)
```

```
39.8
39.8
```

Look at the beauty of the code, you can even compose argument thanks to the composition.

In [25]:

```
print_twice("argument "*4)
```

```
argument argument argument argument
argument argument argument argument
```

In [27]:

```
print_twice(math.cos(math.pi))
```

```
-1.0
-1.0
```

In [28]:

```
print_twice(math.tan(math.pi))
```

```
-1.2246467991473532e-16
-1.2246467991473532e-16
```

One more thing - you can use whole variable as an argument too

In [30]:

```
Mia="Oh baby, come over. I am waiting for you."
print_twice(Mia)
```

```
Oh baby, come over. I am waiting for you.
Oh baby, come over. I am waiting for you.
```

variables and parameters are local

when you create variable inside a function, it is local, which means that it only exists inside the function.

In [33]:

```
def mia_twice(part1,part2):
    mia=part1+part2
    print_twice(mia)
```

function takes 2 arguments, add them and print the final result. I am going to give it two arguments in form of line1 and line2. Let's see

In [36]:

```
line1="Baby, come over."
line2="Babeeee, I'm waiting here."
mia_twice(line1,line2)
```

```
Baby, come over.Babeeee, I'm waiting here.
Baby, come over.Babeeee, I'm waiting here.
```

See mia was used within the function. If I try to call the mia here and it will say there is no mia or mia is not defined.

In [39]:

```
print(mia)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
 last)
<ipython-input-39-d9b7898c4faa> in <module>
----> 1 print(mia)

NameError: name 'mia' is not defined
```

Parameters are also local. For example, outside print_twice, there is not sunch thing like line1 and line2

fruitful and void functions

Those functions who returns a value is called fruitful functions and other functins whose just only prints something blah blah are called void. This is just my notation not documented officially. You can call whatever you want to call.

When you call a function in interactive mode, Python displays the result. But in script, if you call a fruitful fucntion all by itself. The return value is lost forever.

In [41]:

```
math.sqrt(5)
```

Out[41]:

```
2.23606797749979
```

This script computes the sqare root of 5, but since it doesn't store or display the result, it is not very useful. Void functions might display something on screen or have some other effect , but they don't have a return value. If you assign the result to a variable, you get a special value called none.

In [42]:

```
result=print_twice("bing")
```

```
bing
bing
```

In [43]:

```
print(result)
```

```
None
```

In [44]:

```
print(type(None))
```

```
<class 'NoneType'>
```

The value None is not the same as the string 'None'. It is a special value that has its own type and that is NoneType

the functions we have written so far are all void. We will start writing fruitful functions in few lectures.

Debugging

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming. In some ways debugging is like detective work. You are confronted with clues and you have to infer the processes and events that led to the results you see. Debugging is also like an experimental science. Once you have an idea about what is going wrong, you modify your program and try again. If your hypothesis was correct, you can predict the result of the modification, and you take a step closer to a working program. If your hypothesis was wrong, you have to come up with a new one. As Sherlock Holmes pointed out, “When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” (A. Conan Doyle, The Sign of Four). For some people, programming and debugging are the same thing. That is, programming is the process of gradually debugging a program until it does what you want. The idea is that you should start with a working program and make small modifications, debugging them as you go. For example, Linux is an operating system that contains millions of

lines of code, but it started out as a simple program Linus Torvalds used to explore the Intel 80386 chip. According to Larry Greenfield, “One of Linus’s earlier projects was a program that would switch between printing AAAA and BBBB. This later evolved to Linux.” (The Linux Users’ Guide Beta Version 1).

In [ ]:

```
1
```