**Python**

**How python is interpreted?**

**Ans.** Interpreted in simple terms means running code line by line. It also means that the instruction is executed without earlier compiling the whole program into machine language.

**Python is both compiled as well as an interpreted language**. This means when we run a python code, it is first compiled and then interpreted line by line. The compilation part is mostly hidden from the user. While running the code, Python generates a byte code internally, this byte code is then converted using a python virtual machine (p.v.m) to generate the output.

**Q2 What is if \_\_name\_\_ == "\_\_main\_\_"?**

Like other programming languages, Python too has an **execution entry point** i.e. main. A module is a file containing Python definitions and statements. Every module in python has a special attribute called **\_\_name\_\_** . The value of \_\_name\_\_ attribute is set to **'\_\_main\_\_'** when module run as main program.

When you execute a **Python script** , it is treated as the main and its \_\_name\_\_ attribute is set to **"\_\_main\_\_"** . If you import this script as a module in another script, the \_\_name\_\_ is set to the name of the script/module.

By doing the main check, you can have that code only **execute** when you want to run the module as a program and not have it execute when someone just wants to import your **module** and call your functions themselves. Consider the following code for better understanding, it checks if a module is being imported or not.

**Python Example**

print "program started"

if \_\_name\_\_ == "\_\_main\_\_":

print "This is from main module"

else:

print "This is from imported module"

**What is lambda in Python?**

Lambda, the 11th letter of the **Greek alphabet** , is the symbol for wavelength . Lambda comes from the **Lambda Calculus** and refers to anonymous functions in programming.

In Python, Lambda is an **expression** . Lambda's body is a single expression, not a block of statements. Because it is limited to an expression, a **lambda** is less general than a **def** you can only squeeze so much logic into a lambda body without using statements such as if. This is not exactly the same as lambda in functional programming languages, but it is a very powerful concept that's well integrated into **Python** and is often used in conjunction with typical functional concepts like **map()** , **filter()** and **reduce()** . Moreover, Lambda can be used wherever function objects are required.

The general syntax of a lambda function is quite simple:

**lambda argument\_list: expression**

The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments.

**def square\_root(x): return math.sqrt(x)**

**square\_root = lambda x: math.sqrt(x)**

**Lambda in Conditional expressions:**

result = lambda x: "Bigger than 100" if x > 100 else "Smaller than 100"

print(result(99))

**Map example using lambda**

My\_list = [2,3,4,5]

Square = map(lambda x: x\*x, my\_list)

print(list(squared))

[4,9,16,25]

### Filter example using lambda

my\_list = [1, 3,5, 7, 9, 11, 13, 15]

new\_list = list(filter(lambda x: (x%3 == 0) , my\_list))

print(new\_list)

### Reduce example using lambda

from functools import reduce

result = reduce((lambda x, y: x \* y), [1, 2, 3, 4,5])

print(result)

ans = 120

**Runtime vs Compile time**

Runtime and compile time are **programming terms** that refer to different stages of software program development. **Compile-time** is the instance where the code you entered is converted to executable while **Run-time** is the instance where the executable is running. The terms "runtime" and "compile time" are often used by programmers to refer to different types of errors too.

Compile-time checking occurs during the compile time. **Compile time errors** are error occurred due to typing mistake, if we do not follow the proper **syntax** and **semantics** of any programming language then compile time errors are thrown by the compiler. They wont let your program to execute a single line until you remove all the syntax errors or until you debug the compile time errors. The following are usual compile time errors:

* Syntax errors
* Typechecking errors
* Compiler crashes (Rarely)

Run-time type checking happens during run time of programs. **Runtime errors** are the errors that are generated when the program is in running state. These types of errors will cause your program to behave unexpectedly or may even kill your program. They are often referred as **Exceptions** . The following are some usual runtime errors:

* Division by zero
* Dereferencing a null pointer
* Running out of memory