

18CSC207J – Advanced Programming Practice

Course Content

- Declarative Programming Paradigm
- Imperative Programming Paradigm

Unit 3

- Parallel Programming Paradigm
- Concurrent Programming paradigm
- Functional Programming Paradigm

Unit 4

- Logic Programming Paradigm
- Dependent Type Programming Paradigm
- Network Programming Paradigm

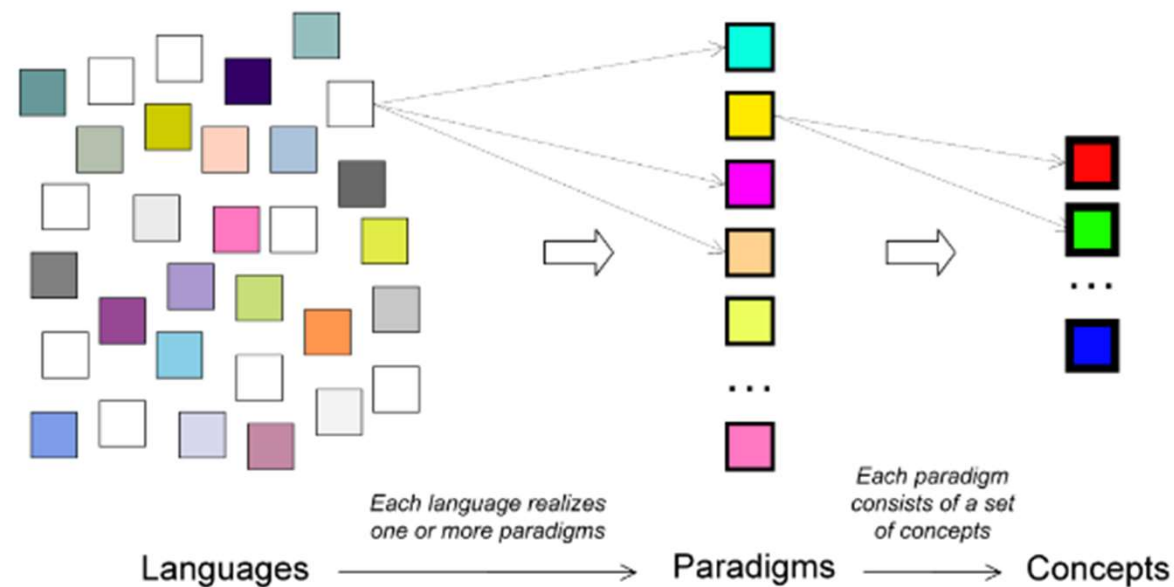
Introduction to Programming Paradigm

Introduction to Programming Paradigm

Hundreds of programming languages are in use...



So many, how can we understand them all?



Key insight:

Languages are based on paradigms, and there are many fewer paradigms than languages

We can ***understand many languages by learning few paradigms***

What is a Programming Paradigm?

- A programming paradigm is an approach to programming a computer based on a coherent set of principles or a mathematical theory
- A program is written to solve problems
 - Any realistic program needs to solve different kinds of problems
 - Each kind of problem needs its own paradigm
 - So we need multiple paradigms and we need to combine them in the same program

How can we study multiple paradigms? How can we combine paradigms in a program?

How can we study multiple paradigms without studying multiple languages (since most languages only support one, or sometimes two paradigms)?

- Each language has its own syntax, its own semantics, its own system, and its own quirks
 - We could pick python language and structure our course around them
- Each paradigm is a different way of thinking
 - How can we combine different ways of thinking in one program?
- We can do it using the concept of a kernel language
 - Each paradigm has a simple core language, its kernel language, that contains its essential concepts
 - Every practical language, even if it's complicated, can be translated easily into its kernel language
 - Even very different paradigms have kernel languages that have much in common; often there is only one concept difference
- We start with a simple kernel language that underlies our first paradigm, functional programming
 - We then add concepts one by one to give the other paradigms

Introduction to Programming Languages

- A ***programming language is an artificial language*** designed to communicate
 - instructions to a machine, e.g., computer
- The earliest programming languages preceded the invention of the computer
 - e.g., used to direct the behavior of machines such as Jacquard looms and player pianos.
- “*Programming languages are the **least usable, but most powerful human-computer interfaces** ever invented*”

Introduction to Programming Languages

the program

- Simplifies program development

Machine code

```
8B542408 83FA0077 06B80000 0000C383
C9010000 008D0419 83FA0376 078BD98B
B84AEBF1 5BC3
```

Assembly language

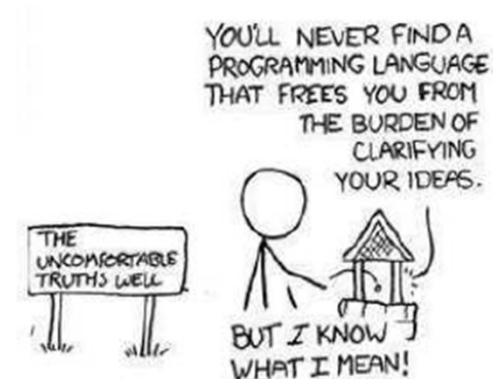
```
mov edx, [esp+8]
cmp edx, 0
ja @f
mov eax, 0
ret
```

High-level language

```
unsigned int fib(unsigned int n) {
```

Introduction to Programming Languages

- Programming languages can be categorized into **programming paradigms**
- Meaning of the word '**paradigm**'
 - “An example that serves as pattern or model”
 - “**Paradigms** emerge as the result of social processes in which people develop ideas and create principles and practices that embody those ideas”
- Programming paradigms are the result of people’s ideas about how computer programs should be constructed
 - Patterns that serves as a “**school of thoughts**” for programming of computers
- Once you have understood the general concepts of programming paradigms, it becomes easier to learn new programming languages



Elements of Programming Language

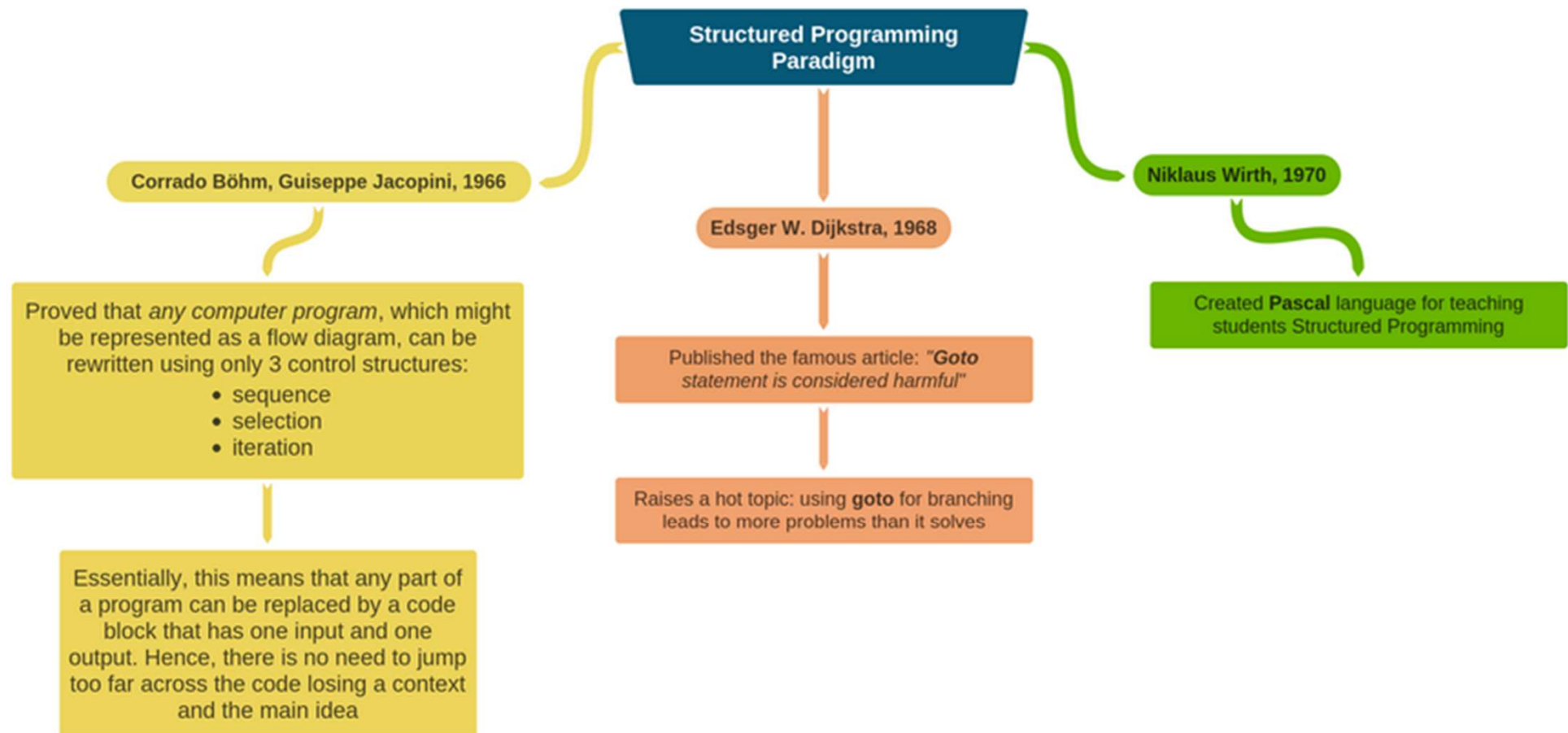
- Programming languages have many similarities with natural languages
 - e.g., they conform to rules for syntax and semantics, there are many dialects, etc.
- We are going to have a quick look at the following concepts
 - Compiled/Interpreted
 - Syntax
 - Semantics
 - Typing

Structured Programming Paradigm

1. Structured Programming Paradigm

- Program is made as a single structure.
- Code will execute the instruction by instruction one after the other.
- It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc.
- The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements.
- Instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are: C, C++, Java, C# ..etc
- The structured program mainly consists of three types of elements:
 - Selection Statements
 - Sequence Statements
 - Iteration Statements

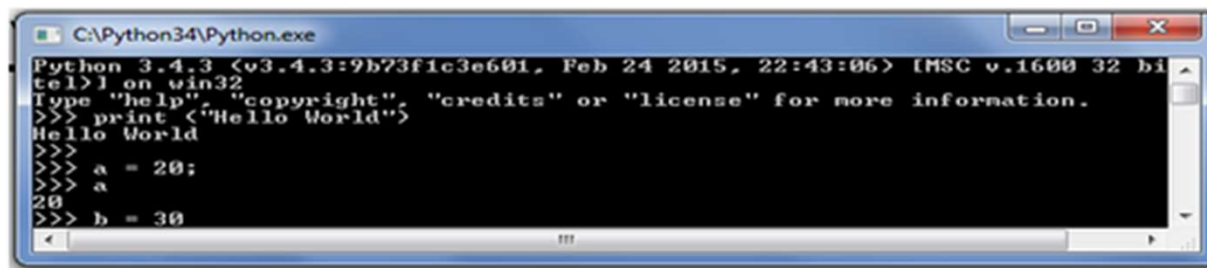
1. Structured Programming Paradigm



1. Structured Programming Paradigm

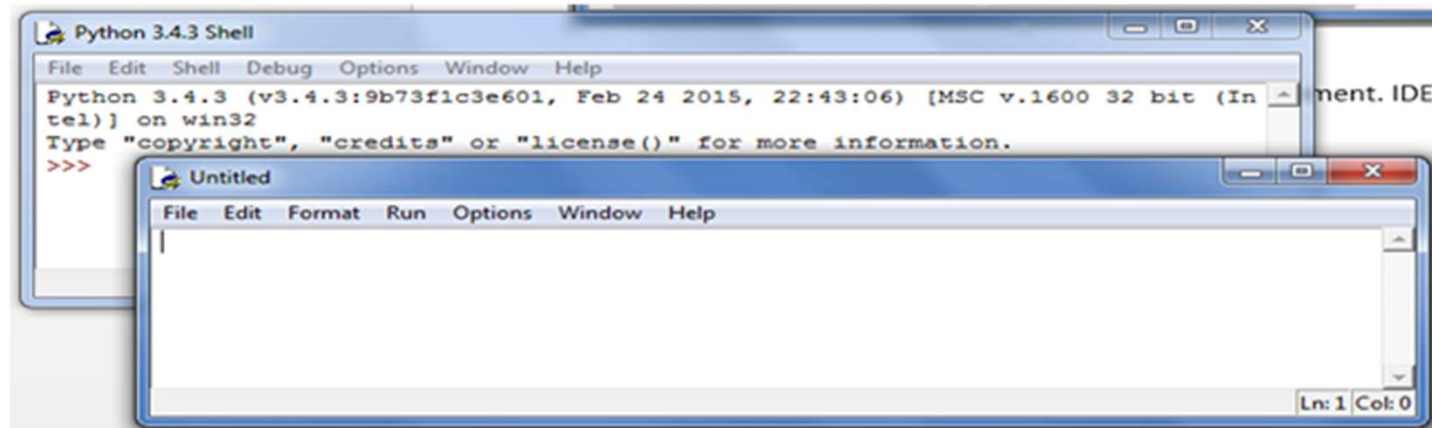
Starting Python

- Python is invoked in two different mode
 - Interactive Mode (By invoking python Interpreter from command prompt)



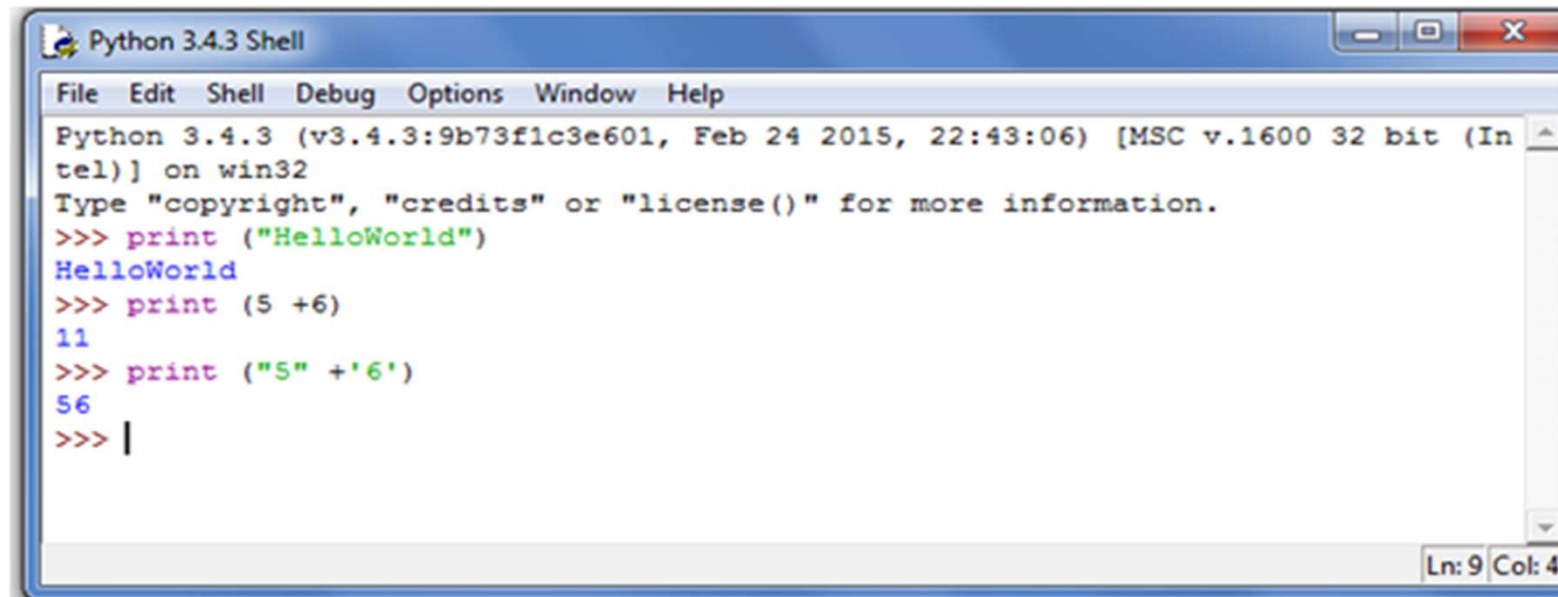
```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print (&quot;Hello World&quot;)
Hello World
>>>
>>> a = 20;
>>> a
20
>>> b = 30
```

- Script Mode (By using IDLE–Interactive Development Environment. IDE used for python)



1. Structured Programming Paradigm

Working with command prompt

A screenshot of a Windows command prompt window titled "Python 3.4.3 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The text inside the window shows the Python version and build information: "Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32". It then prompts the user to type "copyright", "credits", or "license()" for more information. Below this, three lines of Python code are entered and executed: ">>> print ('HelloWorld')", ">>> print (5 +6)", and ">>> print ('5' +'6')". The corresponding outputs "HelloWorld", "11", and "56" are displayed. The prompt ">>> |" is shown at the bottom. The status bar at the bottom right indicates "Ln: 9 Col: 4".

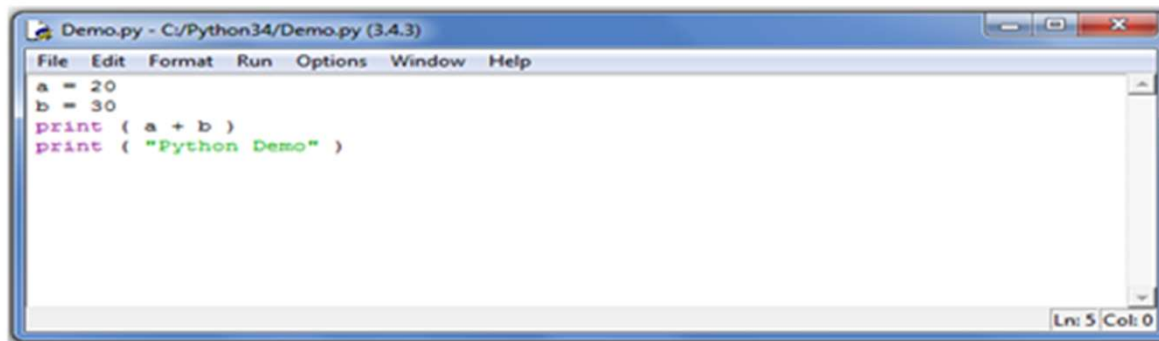
```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ('HelloWorld')
HelloWorld
>>> print (5 +6)
11
>>> print ('5' +'6')
56
>>> |
Ln: 9 Col: 4
```

- Print is the built in function used to print the result back to the console. Console refers to text entry (keyboard) or display device (monitor) of the computer

1. Structured Programming Paradigm

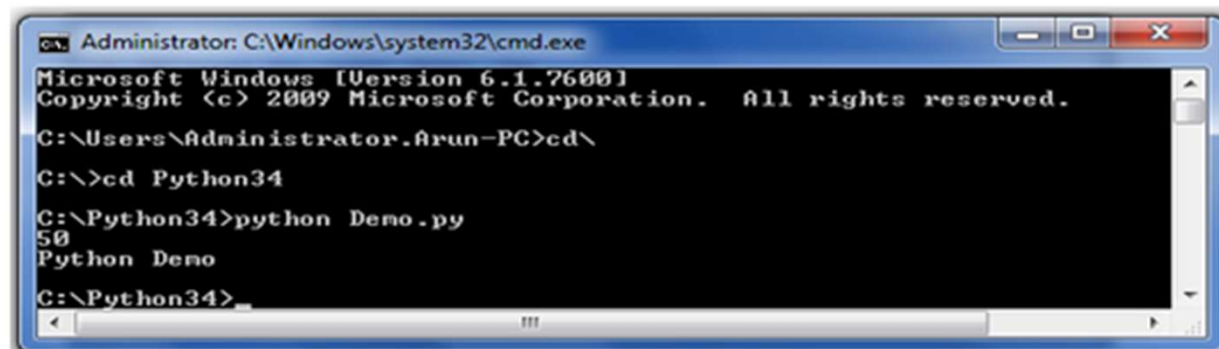
Creating python source file

- Python source file is created using IDLE or notepad and save the contents with .py extension. It reference python source file, script file or module.



```
File Edit Format Run Options Window Help
a = 20
b = 30
print ( a + b )
print ( "Python Demo" )
Ln: 5 Col: 0
```

- Execute the python file using IDLE or execute from command prompt



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator.Arun-PC>cd \
C:\>cd Python34
C:\Python34>python Demo.py
50
Python Demo
C:\Python34>
```

Selection Statement

If else

Syntax:

```
if test expression:  
    statement(s)
```

If else

Syntax:

```
if test expression:  
    Body of if  
else:  
    Body of else
```

If elif else

Syntax:

```
if test expression:  
    Body of if  
elif test expression:  
    Body of elif  
else:  
    Body of else
```

1. Structured Programming Paradigm

Conditional

- A conditional expression evaluates an expression based on a condition.
- Conditional expression is expressed using **if** and **else** combined with expression

Syntax:

expression if Boolean-expression else expression

Example:

Biggest of two numbers

```
num1 = 23
```

```
num2 = 15
```

```
big = num1 if num1 > num2 else num2
```




```
print ( " the biggest number is " , big )
```

Even or odd

```
print ( " num is even " if num % 2 == 0 else " num is odd ")
```

1. Structured Programming Paradigm

Looping

| Loop Type | Description |
|---|--|
| while loop  | Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| for loop  | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| nested loops  | You can use one or more loop inside any another while, for or do..while loop. |

1. Structured Programming Paradigm

For Loop

Syntax:

```
for iterating_var in sequence:  
    statements(s)
```

Example:

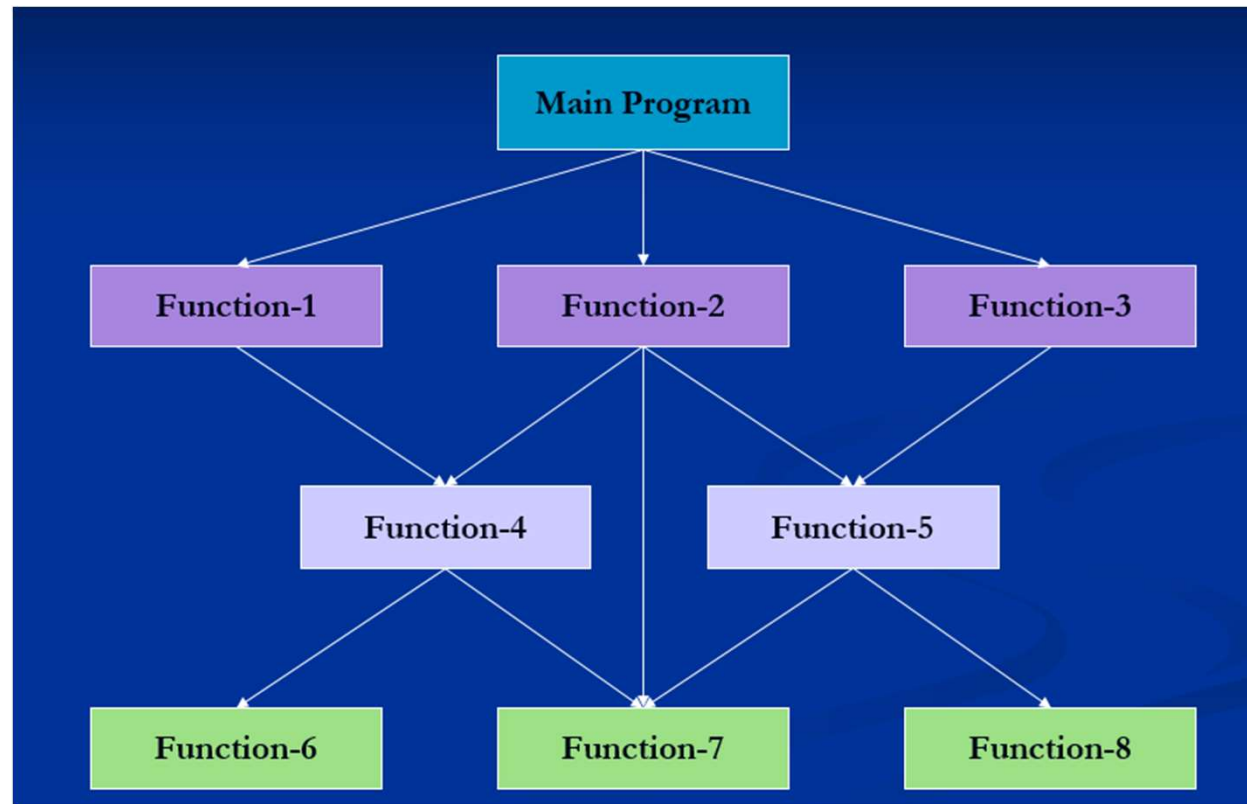
```
for letter in 'Python':    # First Example  
    print 'Current Letter :', letter  
  
fruits = ['banana', 'apple', 'mango']  
for index in range(len(fruits)):  
    print 'Current fruit :', fruits[index]  
print "Good bye!"
```

Procedure Programming Paradigm

Introduction

- High level languages such as COBOL, FORTRAN and C, is commonly known as procedure oriented programming(POP). In the procedure oriented programming, program is divided into sub programs or modules and then assembled to form a complete program. These modules are called functions.
- The problem is viewed as a sequence of things to be done.
- The primary focus is on functions.
- Procedure-oriented programming basically consists of writing a list of instructions for the computer to follow and organizing these instructions into groups known as functions.
- In a multi-function program, many important data items are placed as global so that they may be accessed by all functions. Each function may have its own local data. If a function made any changes to global data, these changes will reflect in other functions. Global data are more unsafe to an accidental change by a function. In a large program it is very difficult to identify what data is used by which function.
- This approach does not model real world problems. This is because functions are action-oriented and do not really correspond to the elements of the problem.

Typical structure of procedure-oriented program



Characteristics of Procedure-Oriented Programming

- Emphasis is on doing things.
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design.

Logical view and Control flow of POP (routine, subroutine and function)

- Procedural programming is a programming paradigm, derived from structured programming, based on the concept of the procedure call. Procedures, also known as routines, subroutines, or functions, simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself.
- procedural languages generally use reserved words that act on blocks, such as if, while, and for, to implement control flow, whereas non-structured imperative languages use goto statements and branch tables for the same purpose.

Note:

- Subroutine:-
- Subroutines; callable units such as procedures, functions, methods, or subprograms are used to allow a sequence to be referred to by a single statement.

Functions in Python

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- There are 2 types of function
 - Built-in function ex. Print()
 - User defined function -User can create their own functions.

Defining a Function

- Function blocks begin with the keyword `def` followed by the function name and parentheses `(())`.
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon `(:)` and is indented.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

```
def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]
```

Function Arguments

- You can call a function by using the following types of formal arguments –
- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Function Arguments

We want to make some of its parameters as optional and use default values if the user does not want to provide values for such parameters.

Example:

```
def say(message, times = 1):  
    print(message * times)
```

```
say('Hello')
```

```
say('World', 5)
```

Note : Default parameters placed at the end of the parameter list.

Keyword Arguments

If you have some functions with many parameters and you want to specify only some of them, then you can give values for such parameters by naming them - this is called keyword arguments. By specifying the name of the parameter we can substitute the value.

Advantages

one, using the function is easier since we do not need to worry about the order of the arguments.

we can give values to only those parameters which we want, provided that the other parameters have default argument values.

Example:

```
def func(a, b=5, c=10):
```

```
    print 'a is', a, 'and b is', b, 'and c is', c
```

```
func(3, 7)
```

```
func(25, c=24)
```

```
func(c=50, a=100)
```

Variable length Parameter

- Python allows us to create functions that can take multiple arguments. So, let's create multi-argument functions.
- In Python, by adding * and ** (one or two asterisks) to the head of parameter names in the function definition, you can specify an arbitrary number of arguments (variable-length arguments) when calling the function.
- By convention, the names *args (arguments) and **kwargs (keyword arguments) are often used, but as long as * and ** are headed, there are no problems with other names. The sample code below uses the names *args and **kwargs.

Example:

```
def my_sum(*args):  
    return sum(args)
```

```
print(my_sum(1, 2, 3, 4))
```

10

```
print(my_sum(1, 2, 3, 4, 5, 6, 7, 8))
```

36

```
def demonstrate_args(arg_1, *argv):  
    print("Argument one-", arg_1)  
    for arg in argv:  
        print("Other arguments-", arg)  
  
demonstrate_args('Hello', 'We', 'are', 'Studytonight')
```


Anonymous function

Usually in any programming function declaration should have a valid identifier as name to be invoked, but python supports user defined function to be defined without any name. Such function are called as anonymous function or lambda function.

Syntax:

```
lambda arguments: expression
```

Example:

```
double = lambda x: x * 2  
print(double(5))
```

Lambda functions are used along with built-in functions like filter(), map() etc

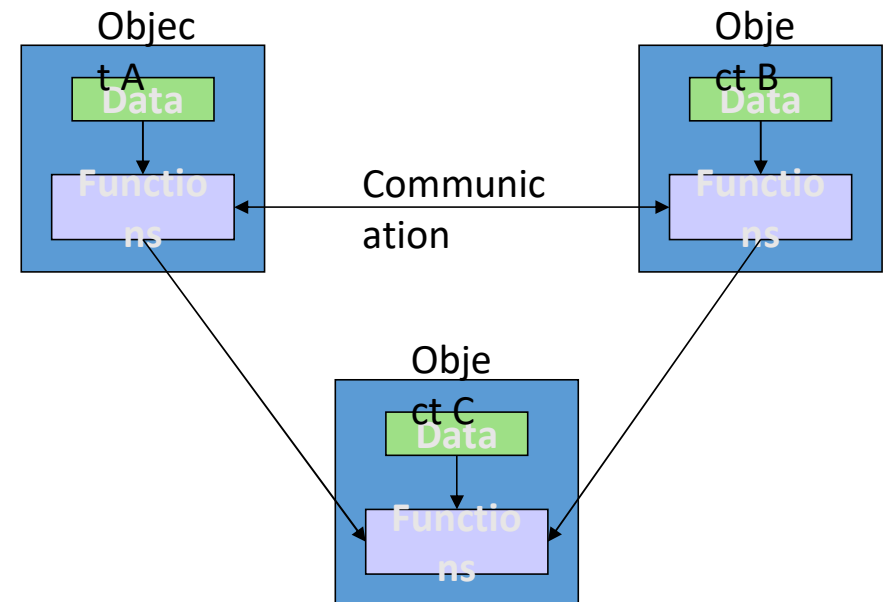
Example:

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]  
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
```

Object Oriented Programming Paradigm

Introduction

- OOP treat data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions.
- OOP allows decomposition of a problem into a number of entities called objects and then build data functions around these objects.
- The data of an object can be accessed only by the functions associated with that object.
- Functions of one object can access the functions of another objects



Characteristics

- Emphasis is on data rather than procedure.
- Programs are divided into objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and can not be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be added easily whenever necessary.
- Follows bottom-up approach in program design.

Basic Concepts of Object-Oriented Programming

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

Classes in Python

- Defining a class is simple, all you have to do is use the keyword `class` followed by the name that you want to give your class, and then a colon symbol `:`. It is standard approach to start the name of class with a capital letter and then follow the camel case style.
- The class definition is included, starting from the next line and it should be indented, as shown in the code below. Also, a class can have variables and member functions in it.

```
class MyClass:
    # member variables
    variable1 = something
    variable2 = something

    # member functions
    def function1(self, parameter1, ...):
        self.variable1 = something else
        # defining a new variable
        self.variable3 = something
        function1 statements...

    def function2(self, parameter1, ...):
        self.variable2 = something else
        function2 statements...
```

Classes in Python

```
class Apollo:
    # define a variable
    destination = "moon"

    # defining the member functions
    def fly(self):
        print "Spaceship flying..."

    def get_destination(self):
        print "Destination is: " + self.destination

# 1st object
objFirst = Apollo()
# 2nd object
objSecond = Apollo()
```

Classes

Note:

- We add the self parameter when we define a member function, but do not specify it while calling the function.
- When we called get_destination function for objFirst it gave output as Destination is: mars, because we updated the value for the variable destination for the object objFirst
- To access a member function or a member variable using an object, we use a dot . symbol.
- And to create an object of any class, we have to call the function with same name as of the class.

```
# Parent class
class Parent:
    # class variable
    a = 10;
    b = 100;
    # some class methods
    def doThis()
    def doThat()

# Child class inheriting Parent class
class Child(Parent):
    # child class variable
    x = 1000;
    y = -1;
    # some child class method
    def dowhat()
    def doNotDoThat()
```


Constructor and Destructor

```
class Example:
    def __init__(self):
        print "Object created"

    # destructor
    def __del__(self):
        print "Object destroyed"

# creating an object
myObj = Example()
# to delete the object explicitly
del myObj
```

Note:

Like Destructor is counter-part of a Constructor, function `__del__` is the counter-part of function `__new__`. Because `__new__` is the function which creates the object.

`__del__` method is called for any object when the reference count for that object becomes zero.

As reference counting is performed, hence it is not necessary that for an object `__del__` method will be called if it goes out of scope. The destructor method will only be called when the reference count becomes zero.

Function Overloading

- In OOP, it is possible to make a function act differently using function overloading. All we have to do is, create different functions with same name having different parameters. For example, consider a function `add()`, which adds all its parameters and returns the result. In python we will define it as,

```
def add(a, b):  
    return a + b
```

- Python doesn't support method overloading on the basis of different number of parameters in functions.

```
# to add 3 numbers  
def add(a, b, c):  
    return a + b + c  
  
# to add 4 numbers  
def add(a, b, c, d):  
    return a + b + c + d
```

Function Overloading

```
def add(a,b):  
    return a+b  
  
def add(a,b,c):  
    return a+b+c  
  
print add(4,5)
```

- If you try to run the above piece of code, you get an error stating, “TypeError: add() takes exactly 3 arguments (2 given)”. This is because, Python understands the latest definition of method add() which takes only two arguments. Even though a method add() that takes care of three arguments exists, it didn't get called. Hence you would be safe to say, overloading methods in Python is not supported.

Inheritance

```
# Parent class
class Parent:
    # class variable
    a = 10;
    b = 100;
    # some class methods
    def doThis()
    def doThat()

# Child class inheriting Parent class
class Child(Parent):
    # child class variable
    x = 1000;
    y = -1;
    # some child class method
    def doWhat()
    def doNotDoThat()
```

Method overriding

Method overriding is a concept of object oriented programming that allows us to change the implementation of a function in the child class that is defined in the parent class. It is the ability of a child class to change the implementation of any method which is already provided by one of its parent class(ancestors).

Following conditions must be met for overriding a function:

Inheritance should be there. Function overriding cannot be done within a class. We need to derive a child class from a parent class.

The function that is redefined in the child class should have the same signature as in the parent class i.e. same number of parameters.

Method Overriding

```
# parent class
class Animal:

    # properties
    multicellular = True
    # Eukaryotic means Cells with Nucleus
    eukaryotic = True

    # function breath
    def breathe(self):
        print("I breathe oxygen.")

    # function feed
    def feed(self):
        print("I eat food.")

# child class
class Herbivorous(Animal):

    # function feed
    def feed(self):
        print("I eat only plants. I am vegetarian.")
```