Python

Agenda

- Module 1 - Core python
- Module 2 - Collections
- Module 3 - Functions
- Module 4 - Modules
- Module 5 - Input output
- Module 6 - Exception Handling
- Module 7 - OOPs Concept
- Module 8 - Regular Expressions
- Module 9 - Networking
- Module 10 - Multithreading
- Module 11 - GUI
- Module 12 - Database
- Module 13 - Django Framework

TOPS Technologies

Module : 1 (Core Python) Fundamentals

- Introduction of Python
- programing Style
- Conditional Statements
- Looping statements
- Control statements
- String Manipulation , operations and Methods

TOPS Technologies

Introduction of Python

- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
- Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development.

TOPS Technologies

Introduction of Python

- Python supports modules and packages, which encourages program modularity and code reuse.

- The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Why Python ?

- **Designed to be easy to learn and master**
  - Clean, clear syntax
  - Very few keywords

- **Highly portable**
  - Runs almost anywhere - high end servers and workstations, down to windows CE
  - Uses machine independent byte-code

- **Extensible**
  - Designed to be extensible using C/C++,
  - allowing access to many external libraries

Features of Python

- **Clean syntax plus high-level data types**
  - Leads to fast coding (First language in many universities abroad!)
- **Uses white-space to delimit blocks**
  - Humans generally do, so why not the language?
  - Try it, you will end up liking it
- **Uses white-space to delimit blocks**
  - Variables do not need declaration
  - Although not a type-less language

TOPS Technologies

Features of Python and Installation

- Python Productivity
  - **Reduced development time**
    Code is 2-10x shorter than C, C++, Java
  - **Improved program maintenance**
    Code is extremely readable
  - **Less training**
    Language is very easy to learn

Programming Style

- Python programs/modules are written as text files with traditionally a .py extension.
- Each Python module has its own discrete namespace.
- Name space within a Python module is a global one.
- Python modules and programs are differentiated only by the way they are called.

Programming Style

- py files executed directly are programs (often referred to as scripts)
- .py files referenced via the import statement are modules.
- Thus, the same .py file can be a program/script, or a module.

Programming Style

- Python programs/modules are written as text files with traditionally a .py extension.
- Each Python module has its own discrete namespace.
- Name space within a Python module is a global one.
- Python modules and programs are differentiated only by the way they are called.

print() function

- The print function in Python is a function that outputs to your console window whatever you say you want to print out.

- At first blush, it might appear that the print function is rather useless for programming, but it is actually one of the most widely used functions in all of python. The reason for this is that it makes for a great debugging tool.

Refer this example :

1.1.1 Print sample

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.1%20print.py

1.1.2 single quotation and double quotation

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.2%20Single-Double%20Quotation%20print.py

TOPS Technologies

Difference between python2 and Python3

- One difference between Python 2 and Python 3 is the print statement. In Python 2, the “print” statement is not a function, and therefore can be invoked without a parenthesis. However, in Python 3, it is a function, and must be invoked with parentheses.

- **Python 2**

```
print "Hello"
```

- **Python 3**

```
print("Hello")
```

Escape Sequences

Escape Sequence	use
\'	Single quote
\"	Double quote
\\	backslash
\n	New line
\t	tab
\b	backspace

1.1.3 Escape sequence

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.3%20Escape%20sequence.py

end= " "

- The end=' ' is just to say that you want a space after the end of the statement instead of a new line character.

Refer This Example :

1.1.4 end practical

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.4%20end%20function%20.py

sep= " "

- The separator between the arguments to print() function in Python is space by default (softspace feature) , which can be modified and can be made to any character, integer or string as per our choice.

- The 'sep' parameter is used to achieve the same, it is found only in python 3.x or later. It is also used for formatting the output strings.

Refer This Example :

1.1.5 sep practical

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.5%20sep.py

Comments

- A comment is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters.

- **Types of comments :**
  - Single line comment

    Indicate by #

  - Document comment

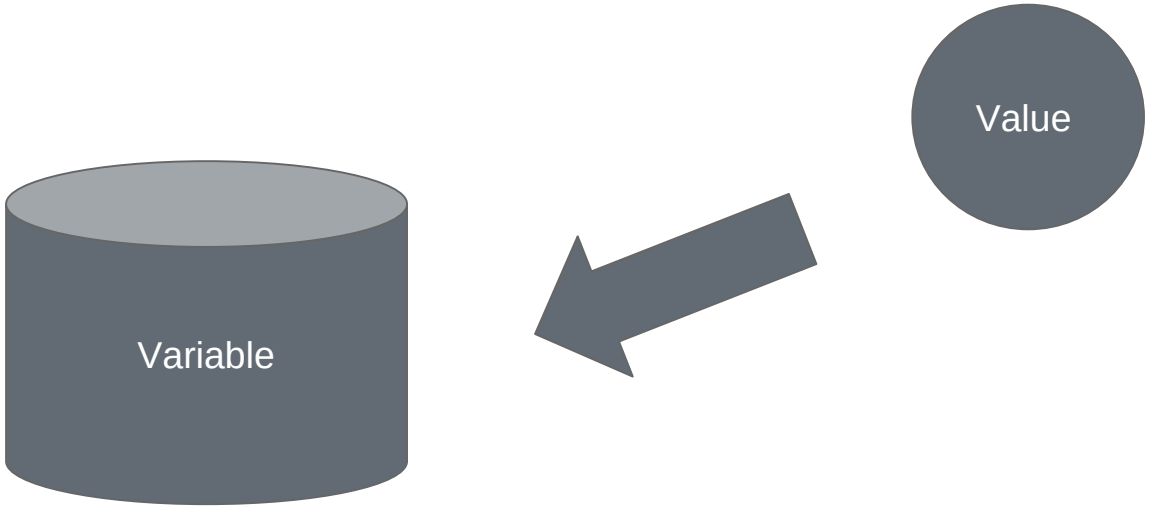
    “””

    statements

    “””

Variables

- Variable : A name which can store a value



- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

E.g. number=20

age = 21

Note : Variables do not need to be declared with any particular type

Variable Declaration Rules :

- A variable can have a short name (like a and b) or a more descriptive name (age,username,product_price).
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number

  1oname = “python”
- A variable name can only contain alphanumeric characters and underscores (A-z, 0-9, and _)

  @name=”Python”

Variable Declaration Rules :

- Variable names are case-sensitive (age, Age and AGE are three different variables)

```
NAME=”python”
print(name)
```

Error : NameError: name 'name' is not defined

Refer This Examples :

1.1.6 Variable sample

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.6%20Variable.py

1.1.7 Sum of two numbers

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.7%20sum%20of%20two%20numbers(variable).py

1.1.8 Swaping of two numbers

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.8%20swaping%20of%20two%20numbers.py

Raw String

- Python raw string is created by prefixing a string literal with 'r' or 'R'. Python raw string treats backslash (\) as a literal character.

Refer this Example :

1.1.9 Raw string

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.9%20raw%20string.py

type()

- Python have a built-in method called as type which generally come in handy while figuring out the type of variable used in the program in the runtime.

Refer this Example :

1.1.10 type function

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.10%20type%20function.py

TOPS Technologies

Casting()

- Convert one data type value into another data type. In python Casting done with function such as int() or float() or str() .

Refer this Example :

1.1.11 Cast

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.11%20casting.py

Taking input from keyboard

- Python provides us inbuilt function to read the input from the keyword.

input() :

- This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list.
- If the input provided is not correct then either syntax error or exception is raised by python.

How input function works:

- When input() function executes program flow will be stopped until the user has given an input.
- The text or message display on the output screen to ask a user to enter input value is optional i.e. the prompt, will be printed on the screen is optional.
- Whatever you enter as input, input function convert it into a string. if you enter an integer value still input() function convert it into a string. You need to explicitly convert it into an integer in your code using typecasting.

Refer this Example :

1.1.12 string input

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.12%20string%20input.py

1.1.13 int input

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.13%20int%20input.py

TOPS Technologies

split()

- Using of split() function at some point, need to break a large string down into smaller chunks, or strings.

Refer this Example :

1.1.14 split

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.14%20split.py

TOPS Technologies

format()

- In Python 3, which allows multiple substitutions and value formatting. This method lets us concatenate elements within a string through positional formatting.

Refer this Example :

1.1.15 format

- https://github.com/TopsCode/Python/blob/master/Module-1/1.1%20Programing%20Style/1.1.15%20format.py

Operators in Python

- To perform specific operations we need to use some symbols .. that symbols are operator

Example :

A + B

Here, + is a operator

A and B is operand

And A+B is expression

Arithmetic Operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

Assignment Operators

Operator	Example
=	a=10
+=	a+=10
*=	a*=10
/=	a/=10
%=	a%=10
**=	a**=10
//=	a//=10

Logical Operators

Operator	name
and	And operator
or	Or operator
not	Not operator

Comparison Operators

Operator	Name
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Identity Operators

Operator	Example
is	A is B
is not	A is not B

Membership Operators

Operator	Example
in	A in student_list
Not in	A not in student_list

Conditional Statements

- Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions
- Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.

Types of conditional Statements

- If .. Statement
- If.. else Statement
- If..Elif..else Statement
- Nested if Statement

If Statements

- It is similar to that of other languages.
- The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
- Syntax :

```
if  condition:
        statements
```

Refer This Example :

If .. else statement

- It is similar to that of other languages.
- It is frequently the case that you want one thing to happen when a condition it true, and something else to happen when it is false.

- For that we have the if else statement.
- Syntax :

```
if  condition:
        statements
    else:
        statement(s)
```

If..elif..else statement

- It is similar to that of other languages.

- The elif is short for else if. It allows us to check for multiple expressions.
- If the condition for if is False, it checks the condition of the next elif block and so on.

- If all the conditions are False, body of else is executed.

- Only one block among the several if...elif...else blocks is executed according to the condition.

If..elif..else statement

- Syntax :

```
If condition:
        statement
Elif condition
        Statement
```

Nested if....else statement

- There may be a situation when you want to check for another condition after a condition resolves to true.
- In such a situation, you can use the nested if construct.
- Syntax :

```
if  condition:
                statements
        if  condition:
                        statements
else:
                statement(s)
```

Refer this Example :

1.2.1 if statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.2%20Conditional%20Statements/1.2.1%20if%20statement.py

1.2.2 if else statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.2%20Conditional%20Statements/1.2.2%20if%20else%20statement.py

1.2.3 elif statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.2%20Conditional%20Statements/1.2.3%20elif%20statement.py

1.2.4 nested if statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.2%20Conditional%20Statements/1.2.4%20nested%20if%20statement.py

Looping Statements

- A loop statement allows us to execute a statement or group of statements multiple times.

- Python programming language provides following types of loops to handle looping requirements.

  - **For Loop**
  - **While Loop**

For Loops

- For loop has the ability to iterate over the items of any sequence, such as a list or a string.
- Syntax : for iterating_var in sequence:
  statements(s)
- If a sequence contains an expression list, it is evaluated first.
- Then, the first item in the sequence is assigned to the iterating variable iterating_var.

- Next, the statements block is executed.
- Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted

Refer this Example :

1.3.1 for loop

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.1%20for%20loop%20with%20string.py

1.3.2 for loop with list

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.2%20for%20loop%20with%20list.py

range() function

- To loop through a set of code a specified number of times, we can use the range() function,
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Refer this Example :

1.3.3 for loop with range

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.3%20for%20loop%20with%20range.py

1.3.4 for loop with decrement range

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.4%20for%20loop%20with%20decrement.py

Nested For loop

- Python programming language allows to use one loop inside another loop.
- Syntax :

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

Refer this Example :

1.3.5 nested for loop

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.5%20nested%20for%20loop.py

1.3.6 pattern 1

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.6%20pattern%201.py

1.3.7 pattern 2

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.7%20pattern%202.py

1.3.8 pattern 3

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.8%20pattern%203.py

1.3.9 pattern 4

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.9%20pattern%204.py

While Loop

- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
- Syntax : while expression:

  statement(s)

1.3.10 while loop

- https://github.com/TopsCode/Python/blob/master/Module-1/1.3%20Looping%20Statement/1.3.10%20while%20loop.py

Control Statements

- Loop control statements change execution from its normal sequence.

- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

- Python supports the following control statements.
  - Break
  - Continue
  - Pass

Break Statements

- It brings control out of the loop and transfers execution to the statement immediately following the loop.
- Syntax :

```
break
```

Refer this Example :

1.4.1 break statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.4%20control%20statament/1.4.1%20break%20statement.py

Continue Statements

- It continues with the next iteration of the loop.
- Syntax :

```
continue
```

Refer this Example :

1.4.2 continue statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.4%20control%20statament/1.4.1%20break%20statement.py

Pass Statements

- The pass statement does nothing.
- It can be used when a statement is required syntactically but the program requires no action.

  Syntax :

  pass

Refer this Example :

1.4.3 pass statement

- https://github.com/TopsCode/Python/blob/master/Module-1/1.4%20control%20statament/1.4.3%20pass%20statement.py

String Manipulation

- Textual data in Python is handled with ”str” objects, or strings. Strings are immutable(fixed/rigid) sequences of Unicode code points.
- String literals are written in a variety of ways:
  - Single quotes: 'allows embedded "double" quotes'
  - Double quotes: "allows embedded 'single' quotes".
  - Triple quoted:

    '''Three single quotes''', """Three double quotes"""

Note : Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.

String Functions

- str.capitalize()

  Return a copy of the string with its first character capitalized and the rest lowercase.

- str.casefold()

  Return a case folded copy of the string. Case folded strings may be used for caseless matching.

- str.center(width[, fillchar])

  Return centered in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).

String Functions

- str.count(sub[, start[, end]])

  Return the number of non-overlapping occurrences of substring sub in the range [start, end].

  Optional arguments start and end are interpreted as in slice notation.

- str.endswith(suffix[, start[, end]])

  Return True if the string ends with the specified suffix, otherwise return False.

  suffix can also be a tuple of suffixes to look for.

  With optional start, test beginning at that position.

  With optional end, stop comparing at that position.

String Functions

- str.find(sub[, start[, end]])

  Return the lowest index in the string where substring sub is found within the slice s[start:end].

  Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.

Note:

The find() method should be used only if you need to know the position of sub. To check if sub is a substring or not, use the ”in”operator:

>>>>>> 'Py' in 'Python' True

String Functions

- str.format(*args, **kwargs)

  Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces{}.

  Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

String Functions

- str.index(sub[, start[, end]])Like find(), but raise ValueError when the substring is not found.
- str.isalnum()

  Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

  A character c is alphanumeric if one of the following returns True:

c.isalpha(), c.isdecimal(), c.isdigit(), or c.isnumeric().

String Functions

- str.isidentifier()

  Return true if the string is a valid identifier according to the language definition

- str.islower()

  Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

- str.istitle()

  Return true if the string is a title cased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones.

  Return false otherwise.

String Functions

- str.isupper()

  Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.

- str.join(*iterable*)

  Return a string which is the concatenation of the strings in the iterable iterable. A TypeError will be raised if there are any non-string values in iterable.

- str.ljust(*width*[, *fillchar*])

  Return the string left justified in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).

String Functions

- str.partition(*sep*)

  Split the string at the first occurrence of sep, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator.

  If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

- str.replace(*old*, *new*[, *count*])

  Return a copy of the string with all occurrences of substring old replaced by new.

  If the optional argument count is given, only the first countoccurrences are replaced.

String Functions

- str.split(*sep=None*, *maxsplit=-1*)

  Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements).

  If maxsplit is not specified or -1, then there is no limit on the number of splits

- str.replace(*old*, *new*[, *count*])

  Return a copy of the string with all occurrences of substring old replaced by new.

  If the optional argument count is given, only the first countoccurrences are replaced.

String Functions

- str.swapcase()
  - Return a copy of the string with uppercase characters converted to lowercase and vice versa.
- str.title()
  - Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

String Slicing

- Like other programming languages, it's possible to access individual characters of a string by using array-like indexing syntax.
- In this we can access each and every element of string through their index number and the indexing starts from 0.
- Python does index out of bound checking.
- So, we can obtain the required character using syntax, string_name[index_position]:
- The positive index_position denotes the element from the starting(0) and the negative index shows the index from the end(-1).

String Slicing

- To extract substring from the whole string then then we use the syntax like
- string_name[beginning: end : step]beginning represents the starting index of string end denotes the end index of string which is not inclusive steps denotes the distance between the two words.

Refer this example :

1.5.1 String demo

- https://github.com/TopsCode/Python/blob/master/Module-1/1.5%20string/1.5.1%20StringDemo.py

1.5.2 String Operations

- https://github.com/TopsCode/Python/blob/master/Module-1/1.5%20string/1.5.2%20StringOperation.py

1.5.3 String slicing

- https://github.com/TopsCode/Python/blob/master/Module-1/1.5%20string/1.5.3%20StringSlicing.py

TOPS Technologies

Module : 2 (Collections)

- **List**

  Operations , Functions and methods

- **Tuple**

  Operations , Functions and methods

- **Dictionaries**

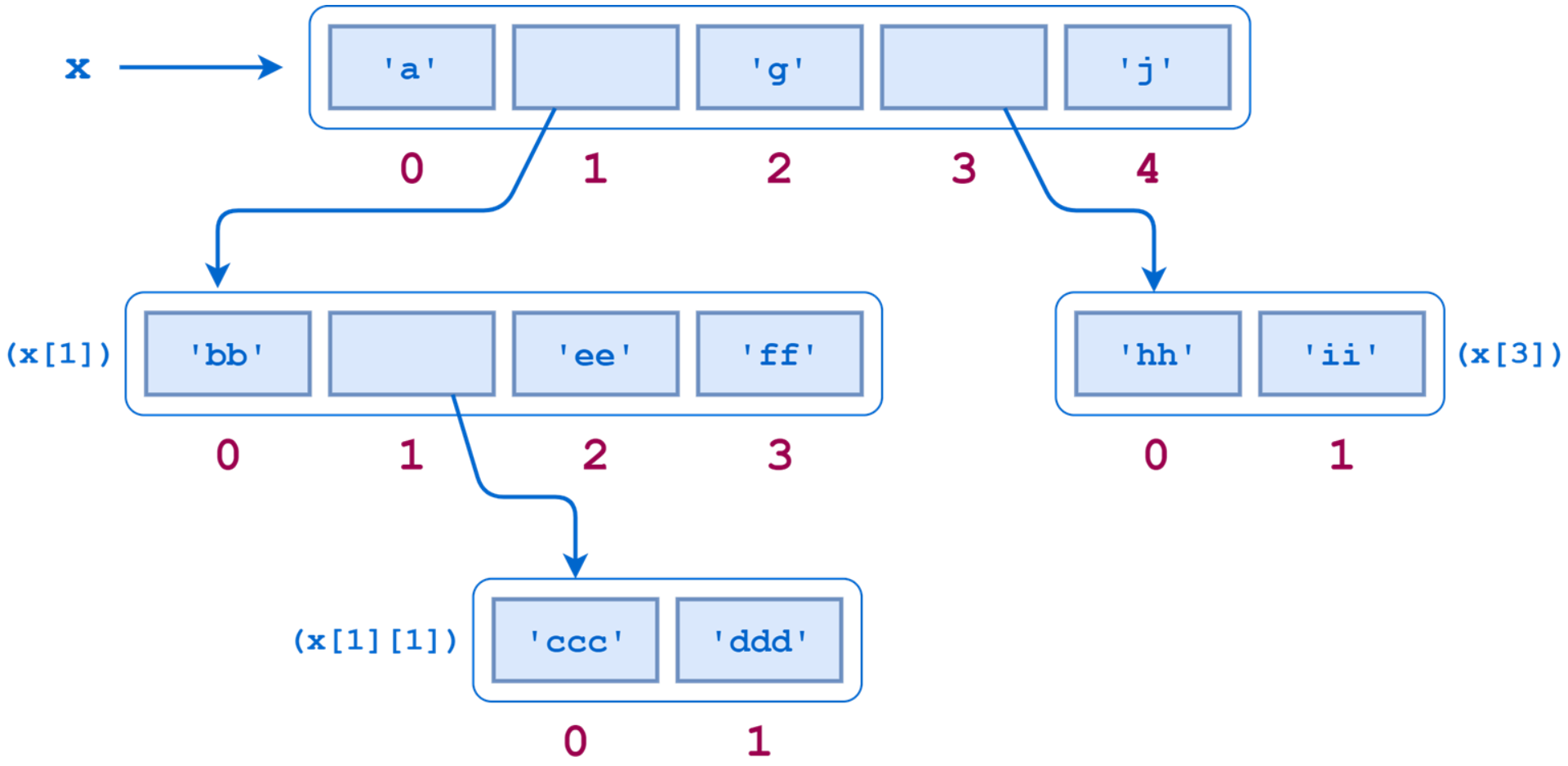
  Operations , Functions and methods

- **Set**

  Operations , Functions and methods

List [ ]

- Introduction
- Accessing list
- Operations
- Working with lists
- Function and Methods


x
'a'
'g'
'j'
0
1
2
3
4
(x[1])
'bb'
'ee'
'ff'
0
1
2
3
'hh'
'ii'
(x[3])
0
1
(x[1][1])
'ccc'
'ddd'
0
1

Built-in Types

Types	Description
numerics	There are three distinct numeric types: *integers*, *floating point numbers*, and *complex numbers*.
Sequences	There are three basic sequence types: lists, tuples, and range objects. Additional sequence types tailored for processing of binary data and text strings.
Mappings	A mapping object maps hashable values to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the *dictionary*.
Classes	Python classes provide all the standard features of Object Oriented Programming.
Instances	instances of user-defined classes.
exceptions	All exceptions must be instances of a class.

Introduction

Python knows a number of compound data types, used to group together other values.

- The most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets.

- Lists might contain items of different types, but usually the items all have the same type.

Accessing List

- Accessing List

  Like strings (and all other built-in sequence type), lists can be indexed and sliced

  Example: fruits[0]

  Example :fruits[-3:-1]

Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content.

Operations

- "**in**" **operator :-** This operator is used to check if an element is present in the list or not. Returns true if element is present in list else returns false.

- "**not in**" **operator :-** This operator is used to check if an element is not present in the list or not.

- Returns true if element is not present in list else returns false.

Operations

- Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

Functions and Methods

Name	Description
len(list)	Gives the total length of the list.
max(list)	Returns item from the list with max value.
min(list)	Returns item from the list with min value.
list(seq)	Converts a tuple into list.

Functions and Methods

Methods	Description
list.append(x)	Add an item to the end of the list. Equivalent to a [len(a):]=[x].
list.extend(L)	Appends the contents of L to list
list.insert(I,x)	Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0,x) inserts at the front of the list,and a.insert(len(a),x) is equivalent to a.append(x).
list.count(obj)	Returns count of how many times obj occurs in list
list.index(obj)	Returns the lowest index in list that obj appears
List.pop(obj=list[-1]	Removes and returns last object or obj from list.

Functions and Methods

Name	Description
list.reverse()	Reverses objects of list in place
list.sort([fun])	Sorts objects of list, use compare **fun** if given
list.remove(obj)	Removes object obj from list

Using Lists as Stacks

- The list methods make it very easy to use a list as a stack, where the last element added is the first element retrieved("last-in,first-out") . To add an item to the top of the stack,use append().
- To retrieve an item from the top of the stack,use pop() without an explicit index.

Using Lists as Stacks

- It is also possible to use a list as a queue,where the first element added is the first element retrieved("first-in, first- out"); however, lists are not efficient for this purpose. While appends and pops from the end of list are fast, doing inserts or pops from the beginning of a list is slow(because all of the other elements have to be shifted by one).

Refer this example :

2.1.1 List as Queue

- https://github.com/TopsCode/Python/blob/master/Module-2/2.1%20List/2.1.1%20ListAsQueue.py

2.1.2 List Demo

- https://github.com/TopsCode/Python/blob/master/Module-2/2.1%20List/2.1.2%20ListDemo.py

2.1.3 List operations

- https://github.com/TopsCode/Python/blob/master/Module-2/2.1%20List/2.1.3%20ListOperation.py

2.1.4 List pattern

- https://github.com/TopsCode/Python/blob/master/Module-2/2.1%20List/2.1.4%20ListPattern.py

TOPS Technologies

Tuple ( )

- Introduction
- Accessing tuples
- Operations
- Working
- Functions and Methods

Introduction

- A tuple is a sequence of immutable Python objects.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
- Eg fruits=(“Mango””,”Banana”,”Oranges”,23,44)
- Eg numbers=(11,22,33,44)
- Eg fruits=“Mango”,”Banana”,”Oranges””

TOPS Technologies

Introduction

- Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once it has been assigned.
- But if the element is itself a mutable data type like list, its nested items can be changed.
- We can also assign a tuple to different values (reassignment).
- Also We cannot delete or remove items from a tuple.
- But deleting the tuple entirely is possible using the keyword del.

Accessing tuples

- There are various ways in which we can access the elements of a tuple.
- We can use the index operator [] to access an item in a tuple.
- Index starts from 0. The index must be an integer.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.
- We can access a range of items in a tuple by using the slicing operator (colon).
- Eg. fruits [2:]

Operations

- With Tuples we can do concate ,repetation,iterations etc...

Functions

Name	Description
len(tuple)	Gives the total length of the tuple.
max(tuple)	Returns item from the tuple with max value.
min(tuple)	Returns item from the tuple with min value.
tuple(seq)	Converts a seq into tuple.

Methods

Method	Description
count(obj)	Returns count of how many times obj occurs in tuple
index(obj)	Returns the lowest index in tuple that obj appears

Refer this example :

2.2.1 add item tuple

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.1%20add_item_tuple.py

2.2.2 convert list tuple

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.2%20convert_list_tuple.py

2.2.3 convert tuple string

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.3%20convert_tuple_string.py

2.2.4 create tuple with numbers

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.4%20create_tuple_withnumbers.py

TOPS Technologies

Refer this example :

2.2.5 create tuple

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.5%20create_tuple.py

2.2.6 Find repeat item tuple

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.6%20find_repaeat_item_tuple.py

2.2.7 slice tuple

- https://github.com/TopsCode/Python/blob/master/Module-2/2.2%20Tuple/2.2.7%20slice_tuple.py

TOPS Technologies

Dictionaries

- Introduction
- Accessing values in dictionaries
- Working with dictionaries
- Properties
- Functions

Introduction

- Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”.

- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys.

- Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.

Introduction

- You can't use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

- The main operations on a dictionary are storing a value with some key and extracting the value given the key.

- Like lists they can be easily changed, can be shrunk and grown ad libitum at run time. They shrink and grow without the necessity of making copies. Dictionaries can be contained in lists and vice versa.

Introduction

- A list is an ordered sequence of objects, whereas dictionaries are unordered sets.
- But the main difference is that items in dictionaries are accessed via keys and not via their position.

Accessing Values

- To access dictionary elements, we can use the familiar square brackets along with the key to obtain its value.
- It is an error to extract a value using a non-existent key.
- We can also create a dictionary using the built-in class dict() (constructor).
- We can test if a key is in a dictionary or not using the keyword in.
- The membership test is for keys only, not for values.

Properties

- **Properties of Dictionaries**
  - Dictionary values have no restrictions.
  - They can be any arbitrary Python object, either standard objects or user-defined objects.
  - However, same is not true for the keys.

- **More than one entry per key not allowed.**
  - Which means no duplicate key is allowed.
  - When duplicate keys encountered during assignment, the last assignment wins.

Properties

- **Keys must be immutable.**
  - Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Methods and Functions

Method	Description
dist.copy()	A dictionary can be copied with the method copy():.
dist.update()	It merges the keys and values of one dictionary into another, overwriting values of the same key.
dist.values()	It returns the list of dictionary dict's values.
dist.keys()	It returns the list of dictionary dict's keys.
dist.items()	It returns the list of dictionary dict's keys,values in tuple pairs.
dist.clear()	Removes all elements of dictionary dict.

Refer this example :

2.3.1 Dictionary example

- https://github.com/TopsCode/Python/blob/master/Module-2/2.3%20Dictionaries/2.3.1%20DictionaryDemo.py

2.3.2 Dictionary method demo

- https://github.com/TopsCode/Python/blob/master/Module-2/2.3%20Dictionaries/2.3.2%20DictionaryMethodDemo.py

TOPS Technologies

Module : 3 (Functions)

- Defining a function
- Calling function
- Types of function
- Anonymous function
- Global and local variables

TOPS Technologies

Functions

- 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.

Defining a Functions

- **Defining a function**
  - Python gives us many built-in functions like print(), etc. but we can also create our own functions.
  - A function in Python is defined by a def statement. The general syntax looks like this:

```
def function-name(Parameter list):
        statements, i.e. the function body
return [expression]
```

Defining a Functions

- The keyword ”def” introduces a function definition.
- It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line, and must be indented.
- The ”return” statement returns with a value from a function.”return” without an expression argument returns None.
- Falling off the end of a function also returns None.

Calling a Functions

- Once function define, we can call it directly or in any other function also.

- Syntax :

```
functionname()
or
functionname(argument)
```

3.1.1 Create function

- https://github.com/TopsCode/Python/blob/master/Module-3/3.1%20Functions/3.1.1%20create%20function.py

Types of Functions

Functions can be of	
Built-in functions	Functions that come built into the Python language itself are called built-in functions and are readily available to us. Eg: input(),eval(),print() etc…
User defined functions	Functions that we define ourselves to do certain specific task are referred as user-defined functions. Eg:checkNoEvenOdd(20)

Function Arguments

- It is possible to define functions with a variable number of arguments.
- The function arguments can be
  - Default arguments values
  - Keyword arguments

Function Arguments

- Default arguments values
  - The most useful form is to specify a default value for one or more arguments.
  - This creates a function that can be called with fewer arguments than it is defined to allow.
  - Eg. def employeeDetails(name,gender='male',age=35)
  - This function can be called in several ways:
  - giving only the mandatory argument : employeeDetails("Ramesh")
  - giving one of the optional arguments: employeeDetails("Ramesh",'Female')
  - or even giving all arguments : employeeDetails("Ramesh",'Female',31)

Function Arguments

- Note : The default value is evaluated only once. This makes a difference when the default is a mutable object such as a list, dictionary, or instances of most classes. For example, the following function accumulates the arguments passed to it on subsequent calls:

Function Arguments

- **Keyword Arguments**
  - Functions can also be called using keyword arguments of the form kwarg=value.
  - For instance, the following function:
  - def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
  - accepts one required argument (voltage) and three optional arguments (state, action, and type).

Function Arguments

def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):	
parrot(1000)	*positional argument*
parrot(voltage=1000)	*keyword argument*
parrot(voltage=1000000, action='VOOOOOM')	*keyword arguments*
parrot(action='VOOOOOM', voltage=1000000)	*keyword arguments*
parrot('a million', 'bereft of life', 'jump')	*positional arguments*
parrot('a thousand', state='pushing up the daisies')	*positional, 1 keyword*

Function Arguments

- **Arbitrary Argument Lists**
  - These arguments will be wrapped up in a tuple . Before the variable number of arguments, zero or more normal arguments may occur.

  - Eg : def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))

  - Normally, these variadic arguments will be last in the list of formal parameters, because they scoop up all remaining input arguments that are passed to the function.

Function Arguments

- Any formal parameters which occur after the *args parameter are ‘keyword-only’ arguments, meaning that they can only be used as keywords rather than positional arguments.

```
def concat(*args, sep="/"):
return sep.join(args)
```

concat("earth", "mars", "venus") O/P->'earth/mars/venus‘
concat("earth", "mars", "venus", sep=".") O/P->'earth.mars.venus'

Refer this example :

3.1.2 Function with parameter

- https://github.com/TopsCode/Python/blob/master/Module-3/3.1%20Functions/3.1.2%20function%20with%20parameter.py

3.1.3 Function with default value

- https://github.com/TopsCode/Python/blob/master/Module-3/3.1%20Functions/3.1.3%20function%20with%20default%20value.py

3.1.4 Function with return values

- https://github.com/TopsCode/Python/blob/master/Module-3/3.1%20Functions/3.1.4%20function%20with%20return%20values.py

3.1.5 tuple as parameter

- https://github.com/TopsCode/Python/blob/master/Module-3/3.1%20Functions/3.1.5%20tuple%20as%20parameter.py

3.1.6 dict as parameter

- https://github.com/TopsCode/Python/blob/master/Module-3/3.1%20Functions/3.1.6%20dict%20as%20parameter.py

Function Arguments

- **Anonymous functions**
  - In Python, anonymous function is a function that is defined without a name.
  - While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.
  - Hence, anonymous functions are also called lambda functions.
  - A lambda function has the following syntax.
  - lambda arguments: expression

Function Arguments

- Lambda functions can have any number of arguments but only one expression.
- The expression is evaluated and returned.
- Lambda functions can be used wherever function objects are required.
- Python supports a style of programming called functional programming where you can pass functions to other functions to do stuff.

3.2.1 lambda function

- https://github.com/TopsCode/Python/blob/master/Module-3/3.2%20Anonymous%20functions/3.2.1%20lambda%20function.py

Global & Local variables

- **Global Variables**
  - Defining a variable on the module level makes it a global variable, you don't need to global keyword.
  - The global keyword is needed only if you want to reassign the global variables in the function/method.
  - Defining a variable on the module level makes it a global variable, you don't need to global keyword.
  - The global keyword is needed only if you want to reassign the global variables in the function/method.

Global & Local variables

- **Local Variables**
  - If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.
  - Local variables of functions can't be accessed from outside

3.3.1 global variable

- https://github.com/TopsCode/Python/blob/master/Module-3/3.3%20Global%20-%20Local%20Variables/3.3.1%20global%20variable.py

3.3.2 local variable

- https://github.com/TopsCode/Python/blob/master/Module-3/3.3%20Global%20-%20Local%20Variables/3.3.1%20global%20variable.py

3.3.3 global keyword

- https://github.com/TopsCode/Python/blob/master/Module-3/3.3%20Global%20-%20Local%20Variables/3.3.3%20global%20keyword%20use.py

Module : 4 Modules

- Importing Module
- Random Module
- Math Module
- Packages
- Composition

Modules

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended.
- Within a module, the module's name (as a string) is available as the value of the global variable __name__.

Importing Module

- Modules can import other modules.
- It is customary but not required to place all import statements at the beginning of a module (or script, for that matter).
- The imported module names are placed in the importing module's global symbol table.

  Eg : import fibo
- Using the module name we can access functions.

```
fibo.fib(10)
fibo.fib2(10)
```

Importing Module

- There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.

For example:

```
from fibo import fib, fib2
fib(500)
another way to import is
from fibo import *
```

4.1.1 module example

- https://github.com/TopsCode/Python/blob/master/Module-4/4.1%20module/4.1.1%20sample.py

Random Module

- Python offers random module that can generate random numbers.

- These are pseudo-random number as the sequence of number generated depends on the seed.

4.2.1 Random module

- https://github.com/TopsCode/Python/blob/master/Module-4/4.2%20random%20module/4.2.1%20RandomModulesDemo.py

Random Functions

Function	Description
random.randrange(*start*, *stop*[, *step*])	Return a randomly selected element from range(start, stop, step). This is equivalent to choice(range(start, stop, step)), but doesn't actually build a range object.
random.randint(*a*, *b*)	Return a random integer *N* such that a <= N <= b. Alias for randrange(a, b+1).
random.choice(*seq*)	Return a random element from the non-empty sequence *seq*.
random.random()	Return the next random floating point number in the range [0.0, 1.0).
And many more…	

Math Module

- This module is always available.
- It provides access to the mathematical functions defined by the C standard.
- This functions are divided into some categories like Number-theoretic and representation functions, Power and logarithmic functions, Trigonometric functions, Angular conversion, Hyperbolic functions, Special functions.
- Constants
- math.pi : The mathematical constant π = 3.141592..., to available precision.
- math.e : The mathematical constant
- e = 2.718281..., to available precision.,

Math Module

- math.inf : A floating-point positive infinity. (For negative infinity, use -math.inf.) Equivalent to the output of float('inf').

- math.nan : A floating-point “not a number” (NaN) value. Equivalent to the output of float('nan')

Function	Description
math.ceil(*x*)	Return the ceiling of *x*, the smallest integer greater than or equal to *x*. If *x* is not a float, delegates to x.__ceil__(), which should return an Integral value.
math.factorial(*x*)	Return *x* factorial

Math Module

Function	Description
math.floor(*x*)	Return the floor of *x*, the largest integer less than or equal to *x*. If *x* is not a float, delegates to x.__floor__(), which should return an Integral value.
math.trunc(*x*)	Return the Real value *x* truncated to an Integral (usually an integer).
math.pow(*x*,	*y*)Return x raised to the power y.
And so more...	

4.3.1 Math module

- https://github.com/TopsCode/Python/blob/master/Module-4/4.3%20math%20module/4.3.1%20MathModuleDemo.py

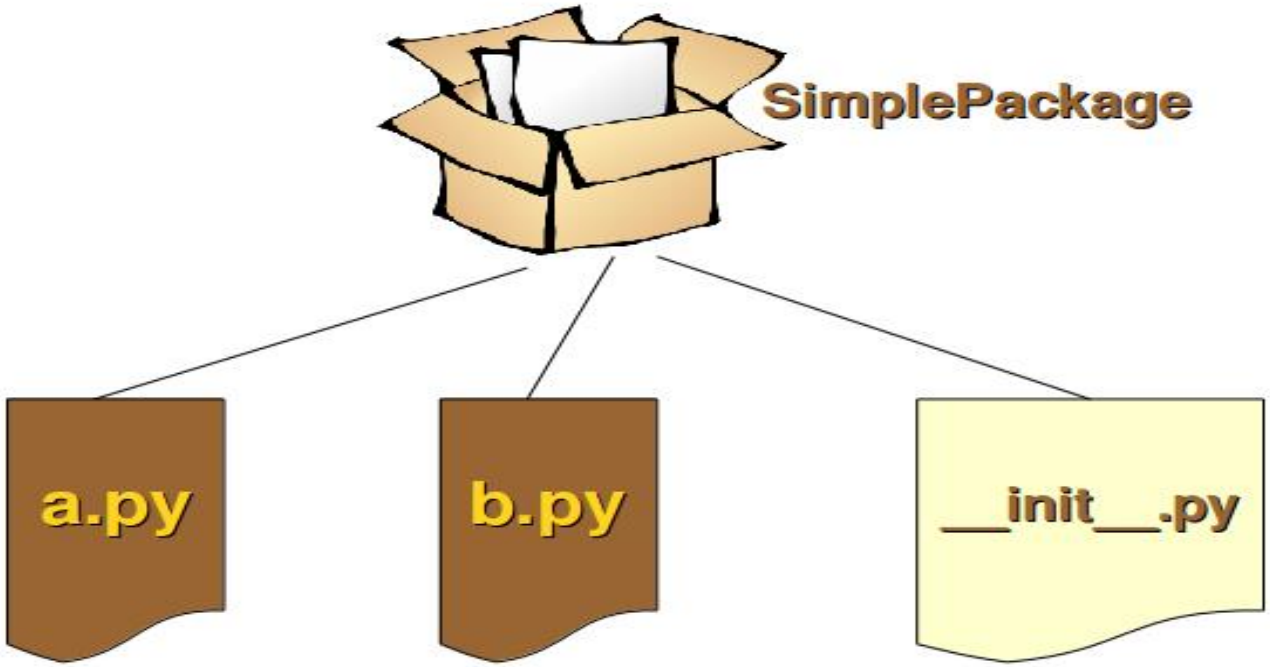
Packages

- Packages are a way of structuring Python's module namespace by using "dotted module names".
- For example, the module name A.B designates a submodule named B in a package named A.
- A package is basically a directory with Python files.

4.4.1 Package example

- https://github.com/TopsCode/Python/blob/master/Module-4/4.4%20packages/4.4.1%20package%20sample.py

Packages



TOPS Technologies

Module : 5 Input-Output

- Printing on Screen
- Open and close file
- Read and write file
- File functions

TOPS Technologies

Printing on screen

- “print()” is use to print on screen.
- print(value, ..., sep=' ', end='\n', file=sys.stdout)
- prints the values to a stream, or to sys.stdout by default.

- **Optional keyword arguments:**

file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.

Reading from keyboard

- To read data from keyboard “input()” is use.
- The input of the user will be returned as a string without any changes.
- If this raw input has to be transformed into another data type needed by the algorithm, we can use either a casting function or the eval function.

Opening and Closing file

- To read data from keyboard “input()” is use.
- “open()” is use to open the file.
- It returns file object.
- syntax
- open(fileName,mode).
- fileName: Name of the file that we wants to open.
- mode: ‘r’ (only for reading), ‘w’ (only for writing), ‘a’ (for append) , r+ (for read and write).
- Normally, files are opened in text mode, that means, you read and write strings from and to the file, which are encoded in a specific encoding.
- If encoding is not specified, the default is platform dependent

.

Opening and Closing file

- **Close the file**

  call f.close() to close it and free up any system resources taken up by the open file.

Reading and writing files

- file.read(size) :This function is use to read a file's contents.
- If size is omitted or negative then the entire content is returned.
- If the end of the file has been reached, f.read() will return an empty string (' ').
- f.readline() reads a single line from the file; a newline character (\n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn't end in a newline.
- For reading lines from a file, you can loop over the file object.
- This is memory efficient, fast, and leads to simple code:

Reading and writing files

- **f.write(string) :** writes the contents of string to the file, returning the number of characters written.
- Other types of objects need to be converted – either to a string (in text mode) or a bytes object (in binary mode) –before writing them.
- **f.tell() :** It returns an integer giving the file object's current position in the file represented as number of bytes from the beginning of the file when in binary mode and an opaque number when in text mode.
- **f.seek():** To change the file object's position.

  f.seek(offset, from_what)

Refer this examples

5.1.1 File example

- https://github.com/TopsCode/Python/blob/master/Module-5/5.1%20File%20-%20input%20output/5.1.1%20FileDemo.py

5.2.1 Create folder

- https://github.com/TopsCode/Python/blob/master/Module-5/5.2%20Folder/5.2.1%20create%20directory.py

5.2.2 delete folder

- https://github.com/TopsCode/Python/blob/master/Module-5/5.2%20Folder/5.2.2%20delete%20directory.py

Module : 6 Exception Handling

- Exception
- Exception handling
- Try , except and finally clause
- User Defined Exceptions

Exception

- In python , there are two distinguishable kinds of errors: syntax errors and exceptions.
- Syntax errors, also known as parsing errors
- Eg: for i in range(1,10)

  print(i)

  Here missing ":" after for is syntax error.

- Exceptions : Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it.
- Errors detected during execution are called exceptions

Exception

- Exceptions handling in Python is very similar to Java.
- But whereas in Java exceptions are caught by catch clauses, we have statements introduced by an "except" keyword in Python.

- Eg : n = int(input("Please enter a number: "))
  Please enter a number: 23.50
  Exception occurs like

  ValueError: invalid literal for int() with base 10: '23.5'

Except clause

- A try statement may have more than one except clause for different exceptions.

- But at most one except clause will be executed.

6.1.1 Exception example 1

- https://github.com/TopsCode/Python/blob/master/Module-6/6.1%20Exception%20Handling/6.1.1%20ExceptionDemo.py

6.1.2 Exception example 2

- https://github.com/TopsCode/Python/blob/master/Module-6/6.1%20Exception%20Handling/6.1.2%20ExceptionDemo2.py

Try …..finally clause

- try statement had always been paired with except clauses. But there is another way to use it as well.

- The try statement can be followed by a finally clause.

- Finally clauses are called clean-up or termination clauses, because they must be executed under all circumstances, i.e. a "finally" clause is always executed regardless if an exception occurred in a try block or not.

6.2.1 Try finally use

- https://github.com/TopsCode/Python/blob/master/Module-6/6.2%20Try%20Finally%20clause/6.2.1%20Try%20finally%20use.py

User defined Exception

- Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

```
class MyNewError(Exception):
        pass
raise MyNewError("Something happened in my program")
```

Module : 7 (Advance Python) OOPs Concept

- Class and object
- Attributes
- Inheritance
- Overloading
- Overriding
- Data hiding

Class And Object

- Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name.

- The class definition looks like :

```
class ClassName:
        Statement 1
        Statement 2
        .....
        Statement N
```

Class And Object

- The statements inside a class definition will usually be function definitions, but other statements are also allowed.

- When a class definition is entered, a new namespace is created, and used as the local scope—thus, all assignments to local variables go into this new namespace.

- In particular, function definitions bind the name of the new function here.

Member methods in class

- The class_suite consists of all the component statements defining class members, data attributes and functions.

- The class attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

- Eg. displayDetails()

Static methods in Python

- Static methods in Python are similar to those found in Java or C++.
- A static method does not receive an implicit first argument.
- To declare a static method, use this idiom:

```
class C:
    def f(arg1, arg2, ...): ...
    f = staticmethod(f)
```

- staticmethod(function) -> method
- Convert a function to be a static method.
- It can be called either on the class (e.g. C.f()) or on an instance
- (e.g. C().f()). The instance is ignored except for its class.

Object

- Class objects support two kinds of operations: attribute references and instantiation.
- Attribute references use the standard syntax used for all attribute references in Python: obj.name
- Valid attribute names are all the names that were in the class's namespace when the class object was created..

Object

- Class objects support two kinds of operations: attribute references and instantiation.
- Class instantiation uses function notation.
- Just pretend that the class object is a parameterless function that returns a new instance of the class.

7.1.1 class and object example1

- https://github.com/TopsCode/Python/blob/master/Module-7/7.1%20class%20and%20object/7.1.1%20ClassDemo.py

7.1.2 class and object example2

- https://github.com/TopsCode/Python/blob/master/Module-7/7.1%20class%20and%20object/7.1.2%20ClassObjectDemo.py

Inheritance

- Python supports inheritance, it even supports multiple inheritance.
- Classes can inherit from other classes.
- A class can inherit attributes and behaviour methods from another class, called the superclass.
- A class which inherits from a superclass is called a subclass, also called heir class or child class.
- Superclasses are sometimes called ancestors as well.

7.2.1 Inheritance

- https://github.com/TopsCode/Python/blob/master/Module-7/7.2%20Inheritance/7.2.1%20InheritenceDemo.py

TOPS Technologies

Overloading

- Python supports operator and function overloading.
- **Method overloading**
  - Overloading is the ability to define the same method, with the same name but with a different number of arguments and types.
  - It's the ability of one function to perform different tasks, depending on the number of parameters or the types of the parameters.
  - Python operators work for built-in classes.
  - But same operator behaves differently with different types.

Overloading

- For example, the + operator will, perform arithmetic addition on two numbers, merge two lists and concatenate two strings.
- This feature in Python, that allows same operator to have different meaning according to the context is called operator overloading.

7.3.1 Operator overloading example1

- https://github.com/TopsCode/Python/blob/master/Module-7/7.3%20Oveloading/7.3.1%20Operator%20overloading.py

7.3.2 Operator overloading example2

- https://github.com/TopsCode/Python/blob/master/Module-7/7.3%20Oveloading/7.3.2%20Operator%20overloading2.py

7.3.3 Operator overloading example3

- https://github.com/TopsCode/Python/blob/master/Module-7/7.3%20Oveloading/7.3.2%20Operator%20overloading3.py

Module : 8 Regular Expression

- Regular Expression
- Match Function
- Search Function
- Matching Vs Searching
- Modifiers
- Patterns

Regular Expression

- Regular expressions (called REs,or regexes,or regexpatterns)are essentially a tiny , highly specialized programming language embedded inside Python and made available through the re module.

- Regular expression patterns are compiled into a series of bytecodes.

Regular Expression

- Python operators work for built-in classes.
- The regular expression language is relatively small and restricted, so not all possible string processing tasks can be done using regular expressions.
- Since regular expressions are used to operate on strings, we'll begin with the most common task: matching characters.

8.1.1 Regular expression

- https://github.com/TopsCode/Python/blob/master/Module-8/8.1%20Regular%20Expression/8.1.1%20regular%20expression.py

Search function

- Scan through a string, looking for any location where this RE matches.

  Syntax :

  re.search(pattern,string, flags=0)

- pattern :This is the regular expression to be matched.
- string :This is the string, which would be searched to match the pattern anywhere in the string.
- flags :You can specify different flags using bitwise OR (|).

Refer this example

8.1.2 Search

- https://github.com/TopsCode/Python/blob/master/Module-8/8.1%20Regular%20Expression/8.1.2%20Search.py

8.1.3 find inter

- https://github.com/TopsCode/Python/blob/master/Module-8/8.1%20Regular%20Expression/8.1.3%20findinter.py

8.1.4 replace

- https://github.com/TopsCode/Python/blob/master/Module-8/8.1%20Regular%20Expression/8.1.4%20replace.py

Match function

- This function determines if the RE matches at the beginning of the string.

  Syntax

  re.match(pattern,string, flags=0)

- pattern :This is the regular expression to be matched.
- string :This is the string, which would be searched to match the pattern at the beginning of string.
- flags :You can specify different flags using bitwise OR (|).

8.1.4 replace

- https://github.com/TopsCode/Python/blob/master/Module-8/8.1%20Regular%20Expression/8.1.4%20replace.py

Match v/s search

- Both apply a pattern. But search attempts this at all possible starting points in the string. Match just tries the first starting point.

8.2.2 Matching Vs searching

- https://github.com/TopsCode/Python/blob/master/Module-8/8.2%20Match/8.2.2%20Matching%20Vs%20Searching.py

Modifiers

- Regular expression literals may include an optional modifier to control various aspects of matching. The modifiers are specified as an optional flag. You can provide multiple modifiers using exclusive OR (|).

Modifiers

Modifier	Description
re.I :IGNORECASE,	Perform case-insensitive matching; character class and literal strings will match letters by ignoring case. For example, [A-Z]will match lowercase letters, too, and Spam will match Spam, spam, or spAM.
re.L LOCALE	
re.M :MULTILINE,	Multi-line matching, affecting ^ and $. When this flag is specified, ^ matches at the beginning of the string and at the beginning of each line within the string, immediately following each newline. Similarly, the $metacharacter matches either at the end of the string and at the end of each line
re.A ASCII	Make \w, \W, \b, \B, \s and \S perform ASCII-only matching instead of full Unicode matching. This is only meaningful for Unicode patterns, and is ignored for byte patterns.

Modifiers

Modifier	Description
re.S DOTALL	Makes the '.' special character match any character at all, including a newline; without this flag, '.' will match anything except a newline.
re.XVERBOSE	When this flag has been specified, whitespace within the RE string is ignored, except when the whitespace is in a character class or preceded by an unescaped backslash; this lets you organize and indent the RE more clearly.

Patterns

- Except for control characters, (+ ? . * ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash.

Description	
|	Alternation, or the “or” operator.
^	Matches at the beginning of lines.
$	Matches at the end of a line,
\A	Matches only at the start of the string.
\Z	Matches only at the end of the string.
\b	Word boundary. This is a zero-width assertion that matches only at the beginning or end of a word.

Patterns

Description	
\w	Matches the word characters.
\W	Matches the nonword characters.
\d	Matches digits.
\D	Matches non digits.
\s	Matches whitespaces.
\S	Matches nonwhitespaces.
\B	Another zero-width assertion, this is the opposite of \b, only matching when the current position is not at a word boundary.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.

Module : 9 Networking

- Socket
- Socket Module
- Methods
- Client and server
- Internet modules

Socket

- Sockets are the endpoints of a bidirectional communications channel.

- Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Socket Module

- To create/initialize a socket, we use the socket.socket() method defined in the Python's socket module.

  Syntax

- sock_obj = socket.socket(socket_family, socket_type, protocol=0)

- **socket_family**: This is either AF_UNIX or AF_INET,
  AF_INET: IP version 4 or IPv4
- AF_UNIX : Unix socket

Socket_type

- **socket_type:** Defines the types of communication between the two end-points. It can have following values.
  - SOCK_STREAM (for connection-oriented protocols e.g. TCP), or
  - SOCK_DGRAM (for connectionless protocols e.g. UDP).
  - SOCK_RAW (For Raw socket)

- **protocol:** This is usually left out, defaulting to 0.

  It's usually used with raw sockets. Like IPPROTO_ICMP, IPPROTO_IP,IPPROTO_RAW,IPPROTO_TCP,IPPROTO_UDP.

Methods

Method	Description
accept()	Accept a new connection
bind(address)	Bind to an address and port
close()	Close the socket
connect(address)	Connect to remote socket
fileno()	Return integer file descriptor
getpeername()	Get name of remote machine
getsockname()	Get socket address as (ipaddr,port)
s.getsockopt(...)	Get socket options
s.listen(backlog)	Start listening for connections

Methods

Method	Description
s.recv(bufsize)	Receive data
recvfrom(bufsize)	Receive data (UDP)
s.send(string)	Send data
s.sendto(string, address)	Send packet (UDP)
s.setblocking(flag)	Set blocking or nonblocking mode
s.setsockopt(...)	Set socket options
s.shutdown(how)	Shutdown one or both halves of connection

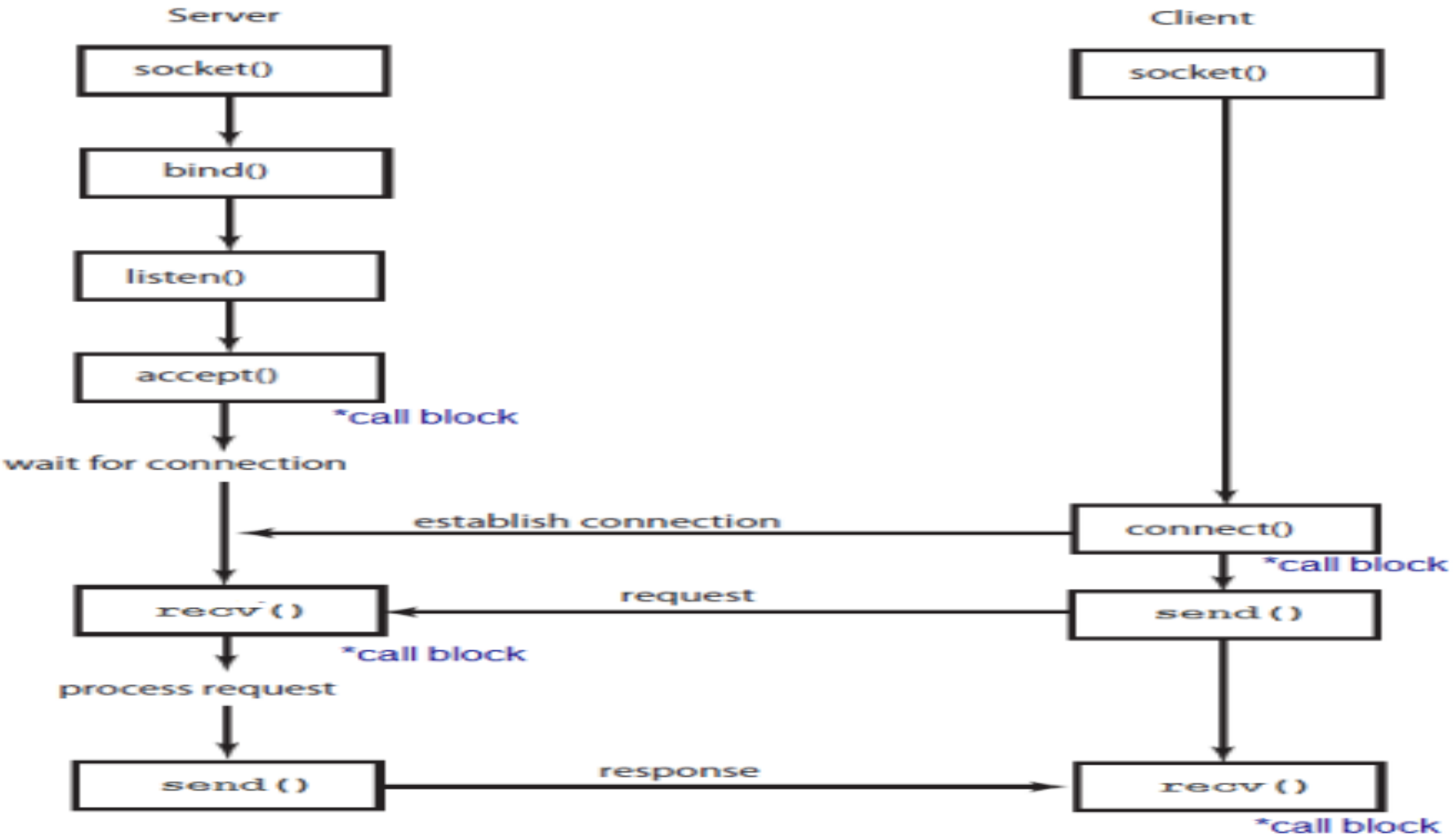
9.1.1 client

- https://github.com/TopsCode/Python/blob/master/Module-9/9.1%20Client/9.1.1%20Client.py

9.2.1 server

- https://github.com/TopsCode/Python/blob/master/Module-9/9.2%20Server/9.2.1%20server.py

How it Works ?


Server
socket()
bind()
listen()
accept()
*call block
wait for connection
recv ()
*call block
process request
send ()
Client
socket()
connect()
*call block
send ()
recv ()
*call block
establish connection
request
response

Module : 10 Multithreading

- Thread
- Starting a thread
- Threading module
- Synchronizing module
- Multithreaded Priority Queue

Thread

- A Thread or a Thread of Execution is defined in computer science as the smallest unit that can be scheduled in an operating system.
- Threads are usually contained in processes.
- More than one thread can exist within the same process.
- Every process has at least one thread, i.e. the process itself.
- A process can start multiple threads.

Starting Thread

- There are two modules which support the usage of threads in Python:
  thread
  &
  threading
- It's possible to execute functions in a separate thread with the module Thread.
- To do this, we can use the function thread.start_new_thread:
- thread.start_new_thread(function, args[, kwargs])

Threading module

- The threading module constructs higher-level threading interfaces on top of the lower level _thread module.
- Creating Thread using threading module.
- Define a new subclass of the Thread class.
- Override the __init__(self [,args]) method to add additional arguments.
- Then, override the run(self [,args]) method to implement what the thread should do when started.

Threading module

- Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls run()method.

Threading module

- The threading module exposes all the methods of the thread module and provides some additional methods:

Method	Description
threading.activeCount():	Returns the number of thread objects that are active.
threading.currentThread():	Returns the number of thread objects in the caller's thread control.
threading.enumerate():	Returns a list of all thread objects that are currently active.

Threading module

- The threading module has the Thread class that implements threading. The methods provided by the Thread class are as follows:

Method	Description
run():	The run() method is the entry point for a thread.
start():	The start() method starts a thread by calling the run method.
join([time]):	The join() waits for threads to terminate.
isAlive():	The isAlive() method checks whether a thread is still executing.
getName():	The getName() method returns the name of a thread.
setName():	The setName() method sets the name of a thread.

Synchronizing threads

- The <*threading*> module has built in functionality to implement locking that allows you to synchronize threads.
- Locking is required to control access to shared resources to prevent corruption or missed data.
- You can call *Lock()* method to apply locks, it returns the new lock object.
- Then, you can invoke the *acquire(blocking)* method of the lock object to enforce threads to run synchronously.

Synchronizing threads

- The optional blocking parameter specifies whether the thread waits to acquire the lock.
  - In case, blocking is set to zero, the thread returns immediately with a zero value if the lock can't be acquired and with a 1 if the lock was acquired.
  - In case, blocking is set to 1, the thread blocks and wait for the lock to be released.
- The release() method of the lock object is used to release the lock when it is no longer required.

Multithreaded Priority Queue

- The Queue module allows you to create a new queue object that can hold a specific number of items.
- Sometimes the processing order of the items in a queue needs to be based on characteristics of those items, rather than just the order they are created or added to the queue.

Multithreaded Priority Queue

Methods	Description
get():	The get() removes and returns an item from the queue.
qsize() :	The qsize() returns the number of items that are currently in the queue.
put():	The put adds item to a queue.
empty():	The empty( ) returns True if queue is empty; otherwise, False.
full():	The full() returns True if queue is full; otherwise, False.

10.1.1 multithreading example 1

- https://github.com/TopsCode/Python/blob/master/Module-10/10.1%20multithreding.py

10.1.2 multithreading example 2

- https://github.com/TopsCode/Python/blob/master/Module-10/10.2%20multithreading2.py

Module : 11 CGI and GUI

- Introduction
- Architecture
- CGI environment variable
- GET and POST methods
- Cookies
- File upload
- Tkinter Programming
- Tkinter Widgets

TOPS Technologies

Introduction

- CGI script is invoked by an HTTP server, usually to process user input submitted through an HTML <FORM> or <ISINDEX> element.

- Most often, CGI scripts live in the server's special cgi-bin directory.

- The HTTP server places all sorts of information about the request (such as the client's hostname, the requested URL, the query string, and lots of other goodies) in the script's shell environment, executes the script, and sends the script's output back to the client.

Introduction

- The script's input is connected to the client too, and sometimes the form data is read this way; at other times the form data is passed via the "query string" part of the URL.

- The output of a CGI script should consist of two sections, separated by a blank line.

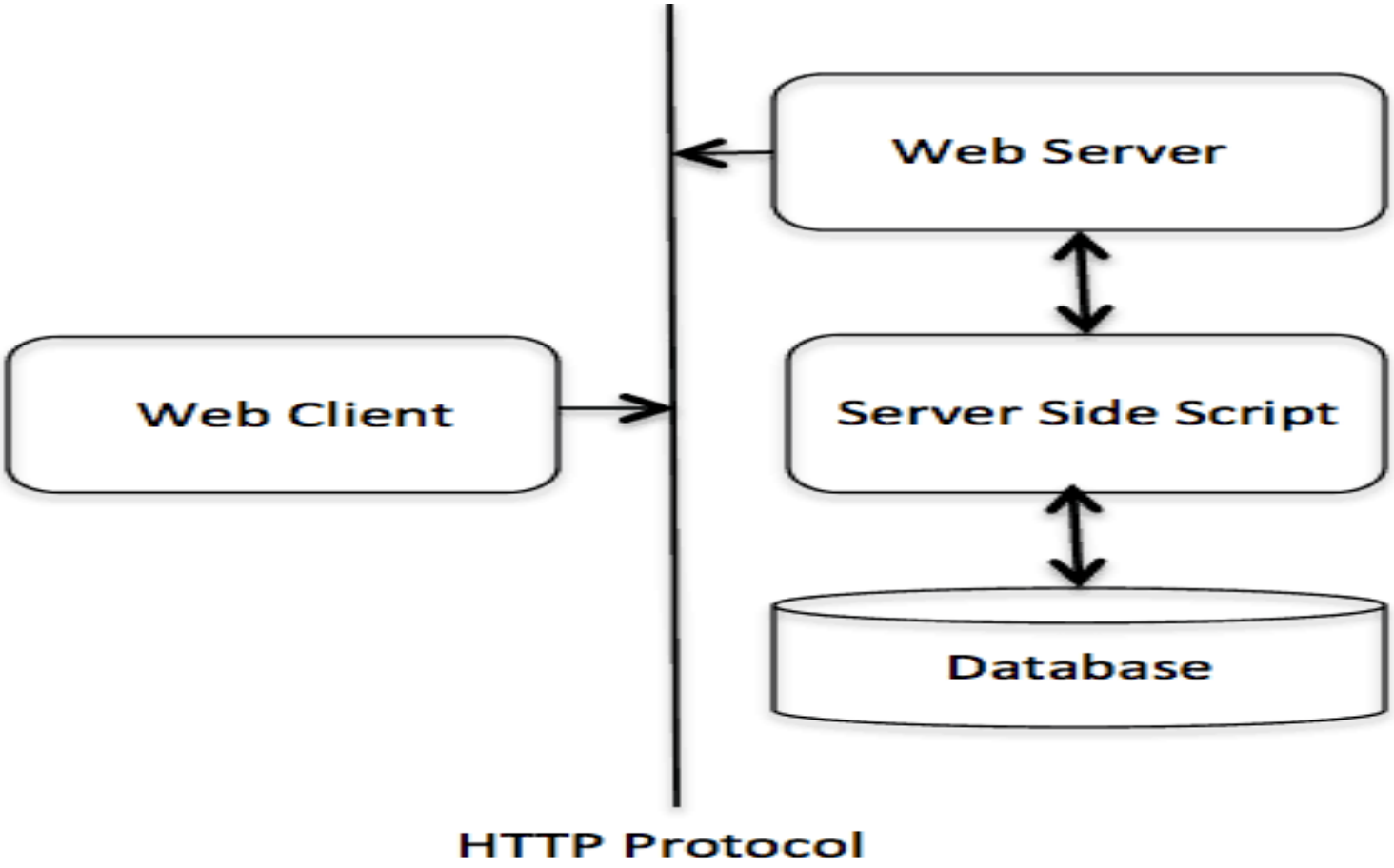
Introduction

- The first section contains a number of headers, telling the client what kind of data is following. Python code to generate a minimal header section looks like this:

```
print ("Content-Type: text/html") # HTML is following
print ()                          # blank line, end of headers
```

- The second section is usually HTML, which allows the client software to display nicely formatted text with header, in-line images, etc. Here's Python code that prints a simple piece of HTML:

Architecture



CGI environment variable

Variables	
CONTENT_TYPE	The data type of the content. Used when the client is sending attached content to the server. For example, file upload.
CONTENT_LENGTH	The length of the query information. It is available only for POST requests.
HTTP_COOKIE	Returns the set cookies in the form of key & value pair.
HTTP_USER_AGENT	The User-Agent request-header field contains information about the user agent originating the request. It is name of the web browser.
SERVER_NAME	The server's hostname or IP Address
SERVER_SOFTWARE	The name and version of the software the server is running.

CGI environment variable

Variables	
PATH_INFO	The path for the CGI script.
QUERY_STRING	The URL-encoded information that is sent with GET method request.
REMOTE_ADDR	The IP address of the remote host making the request. This is useful logging or for authentication.
REMOTE_HOST	The fully qualified name of the host making the request. If this information is not available, then REMOTE_ADDR can be used to get IR address.
REQUEST_METHOD	The method used to make the request. The most common methods are GET and POST.
SCRIPT_FILENAME	The full path to the CGI script.
SCRIPT_NAME	The name of the CGI script.

GET and POST methods

- Browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.
- The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? Character like
- http://www.xyz.com/cgi-bin/hello.py?key1=value1&key2=value2

GET and POST methods

- The GET method is the default method to pass information from browser to web server.
- Never use GET method if you have password or other sensitive information to pass to the server.
- The GET method has size limitation: only 1024 characters can be sent in a request string.

GET and POST methods

- The GET method is the default method to pass information from browser to web server
- The GET method sends information using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable.

POST methods

- A generally more reliable method of passing information to a CGI program is the POST method.
- This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message.
- This message comes into the CGI script in the form of the standard input.

Cookies

- A generally more reliable method of passing information to a CGI program is the POST method.
- HTTP protocol is a stateless protocol.
- For a commercial website, it is required to maintain session information among different pages.
- For example, one user registration ends after completing many pages.
- In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

Cookies - How it Works ?

- Your server sends some data to the visitor's browser in the form of a cookie.
- The browser may accept the cookie.
- If it does, it is stored as a plain text record on the visitor's hard drive.
- Now, when the visitor arrives at another page on your site, the cookie is available for retrieval.

Cookies

- Cookies are a plain text data record of 5 variable-length fields:
- **Expires:** The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain:** The domain name of your site.
- **Path:** The path to the directory or web page that sets the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure:** If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value:** Cookies are set and retrieved in the form of key and value pairs.

File Upload

- To upload a file, the HTML form must have the enctype attribute set to multipart/form-data. The input tag with the file type creates a "Browse" button.

Tkinter

- Tkinter is the standard GUI library for Python.
  - Python when combined with Tkinter provides a fast and easy way to create GUI applications.
  - Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

11.1.1 window creation

- https://github.com/TopsCode/Python/blob/master/Module-11/11.1%20Tkinter%20Programing/11.1.1%20window.py

GUI Programming

- Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –
  - Import the Tkinter module.
  - Create the GUI application main window.
  - Add one or more of the above-mentioned widgets to the GUI application.
  - Enter the main event loop to take action against each event triggered by the user.

GUI Programming

- Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

- E.g.

  Button,Canvas,Checkbutton,Entry,Frame,Label,Listbox,Menubutton,Menu etc…

GUI Programming : Methods

Method	Description
widget.pack(pack_options)	Pack_options can be 'expand','fill', 'side' **expand**:When set to true, widget expands to fill any space not otherwise used in widget's parent. **fill:** Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically). **side:** Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

GUI Programming : Methods

Method	Description
widget.place(place_options)	place_options are anchor,bordermode,height,width,relheight,relwidth,relx,rely,x,y **anchor :** The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget) **bordermode :** INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.

GUI Programming : Methods

Method	Description
widget.place(place_options)	**relheight, relwidth :** Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget. **relx, rely :** Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget. **x, y :** Horizontal and vertical offset in pixels.

Refer this example

11.2.1 Tkinter with widget 1

- https://github.com/TopsCode/Python/blob/master/Module-11/11.2%20Tkinter%20widgets/11.2.1%20Tkinter%20with%20widget1.py

11.2.2 Tkinter with widget 2

- https://github.com/TopsCode/Python/blob/master/Module-11/11.2%20Tkinter%20widgets/11.2.2%20Tkinter%20with%20widget2.py

11.2.3 Messagebox

- https://github.com/TopsCode/Python/blob/master/Module-11/11.2%20Tkinter%20widgets/11.2.3%20Messagebox.py

11.2.4 Tkinter demo

- https://github.com/TopsCode/Python/blob/master/Module-11/11.2%20Tkinter%20widgets/11.2.4%20TkinterDemo.py

11.2.5 Tkinter demo2

- https://github.com/TopsCode/Python/blob/master/Module-11/11.2%20Tkinter%20widgets/11.2.5%20TkinterDemo2.py

Module : 12 Database

- Introduction
- Connections
- Executing queries
- Transactions
- Handling error

Introduction

- SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language

Connections

- To use the module, you must first create a Connection object that represents the database. Here the data will be stored in the example.db file:

```
import sqlite3
conn=sqlite3.connect('example.db')
```

- You can also supply the special name :memory: to create a database in RAM.

Executing queries

- Once you have a Connection, you can create a Cursor object and call its execute() method to perform SQL commands.

- There are execute(),executemany(),executescript() methods to execute queries.

Transaction

- By default, the sqlite3 module opens transactions implicitly before a Data Modification Language (DML) statement (i.e.INSERT/UPDATE/DELETE/REPLACE), and commits transactions implicitly before a non-DML, non-query statement .

12.1.1 Database demo

- https://github.com/TopsCode/Python/blob/master/Module-12/12.1%20Database/12.1.1%20DatabaseDemo.py

Transaction

- **Atomicity:** Either a transaction completes or nothing happens at all.
- **Consistency:** A transaction must start in a consistent state and leave the system in a consistent state.
- **Isolation:** Intermediate results of a transaction are not visible outside the current transaction.
- **Durability:** Once a transaction was committed, the effects are persistent, even after a system failure.
- The Python DB API 2.0 provides two methods to either commit or rollback a transaction.

Module : 13 Django (Framework)

- Introduction
- Virtual Environment
- Installation
- Migration
- Settings.py
- Creating application
- Models

TOPS Technologies

Designing

- Introduction
- HTML
- CSS
- JavaScript
- Ajax
- Bootstrap

Note : This will cover in the Django framework.

What is Django?

- Django is a free and open source web application framework, written in Python. A web framework is a set of components that helps you to develop websites faster and easier.

- When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc.

Introduction

- The web server reads the letter and then sends a response with a web page.

- But when you want to send something, you need to have some content. And Django is something that helps you create the content.
- It is based on MVT (Model View Template) design pattern.

- Easy to build web application in less time

How it works?

- When a request comes to a web server, it's passed to Django which tries to figure out what is actually requested.

- It takes a web page address first and tries to figure out what to do. This part is done by Django's urlresolver (note that a website address is called a URL – Uniform Resource Locator – so the name urlresolver makes sense).

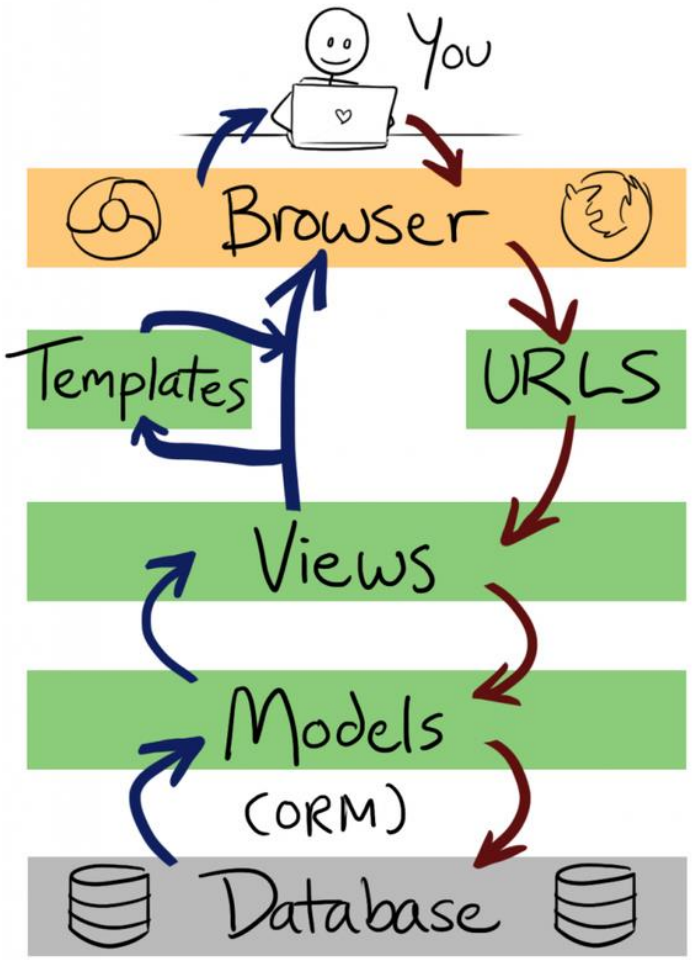
How it works?

- It is not very smart – it takes a list of patterns and tries to match the URL. Django checks patterns from top to bottom and if something is matched, then Django passes the request to the associated function (which is called view).

What is pip ?

- pip is a package-management system used to install and manage software packages written in Python.
- Many packages can be found in the default source for packages and their dependencies — Python Package Index.

How it works?



TOPS Technologies

Django installation

- **Virtual environment**
  - Before we install Django we will get you to install an extremely useful tool to help keep your coding environment tidy on your computer.
  - It's possible to skip this step, but it's highly recommended.
  - So, let's create a virtual environment (also called a virtualenv). Virtualenv will isolate your Python/Django setup on a per-project basis.
  - This means that any changes you make to one website won't affect any others you're also developing

Start the Project

- The first step is to start a new Django project.
- Basically, this means that we'll run some scripts provided by Django that will create the skeleton of a Django project for us.
- This is just a bunch of directories and files that we will use later.
- Here ‘manage.py’ is a script that helps with management of the site. With it we will be able (amongst other things) to start a web server on our computer without installing anything else.

Start the Project

- The settings.py file contains the configuration of your website.
- urls.py file contains a list of patterns used by urlresolver.
- On Windows (again, don't forget to add the period (or dot) '.' at the end.
- Command-line
- (myvenv)C:\Users\Name\djangogirls>django-admin.py startproject mysite.
- Here 'django-admin.py' is a script that will create the directories and files for you.

Create Application

- A model in Django is a special kind of object --it is saved in the database.
- Model in database as a spreadsheet with columns(fields) and rows (data).

- **Creating an application**
  - To create an application we need to run the following command in the console
  - (myvenv)~/djangogirls$ python manage.py startapp blog

Create Application

- You will notice that a new ‘blog’ directory is created and it contains a number of files now.
- The directories and files in our project should look like this:

```
djangogirls
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── db.sqlite3
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Create Application

- After creating an application, we also need to tell Django that it should use it. We do that in the file mysite/settings.py.
- We need to find INSTALLED_APPS and add a line containing ‘blog’ just above.
- e.g.

```
INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        ‘blog’ ]
```

Create Model

- **Creating a blog post model:**
- In the blog/models.py file we define all objects called Models – this is a place in which we will define our blog post
- blog/models.py

```
class Post(models.Model):
    ..........
    ..........
```

Create Model

- **Create tables for models in your database**
- The last step here is to add our new model to our database.
- First we have to make Django know that we have some changes in our model. (We have just created it!)
- Go to your console window and type

  python manage.py makemigrations blog

- Django prepare a migration file for us that we now have to apply to our database. Type

  python manage.py migrate blog

Django Admin

- To add, edit and delete the posts we've just modeled, we will use Django admin.

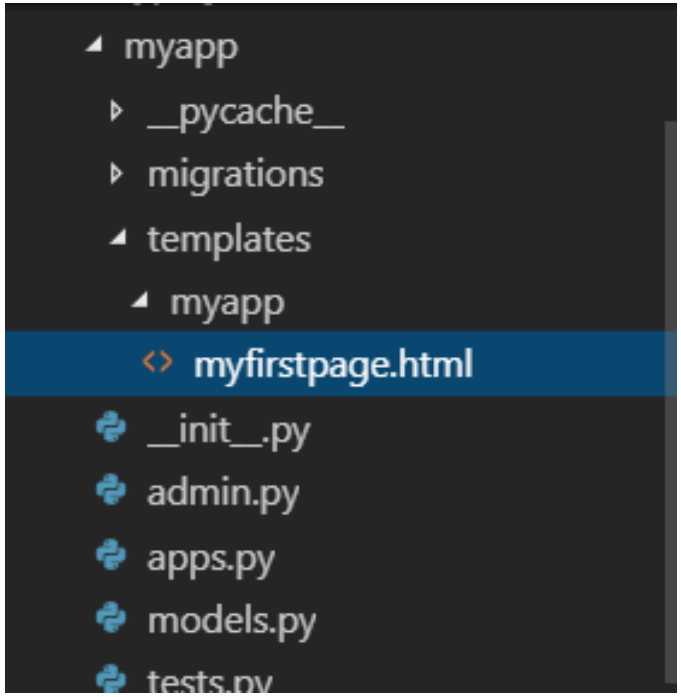
- Let's open the blog/admin.py file and replace its contents with this:

```
blog/admin.py
from django.contrib import admin
from .models import Post
admin.site.register(Post)
```

Django Template

- Django provides a convenient way to generate dynamic HTML pages by using its template system.
- A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.
- In HTML file, we can't write python code because the code is only interpreted by python interpreter not the browser. We know that HTML is a static markup language, while Python is a dynamic programming language.
- Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.

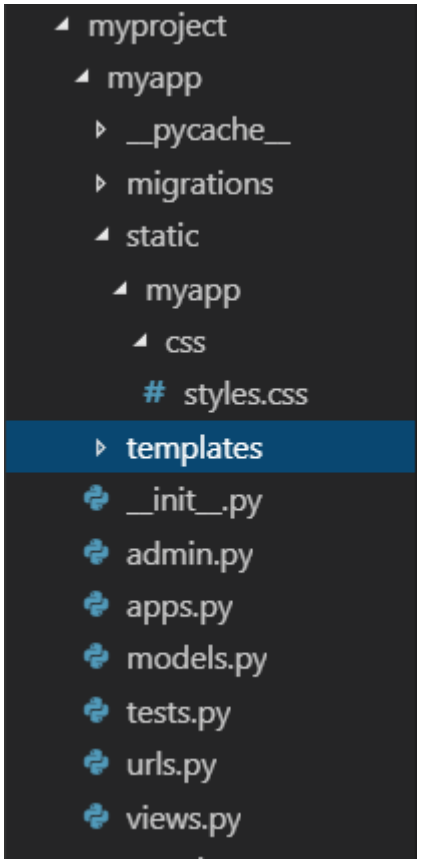
Django Template



Django static files

- To handle and Manage static resource of website like css,JavaScript,images etc..

```
INSTALLED_APPS = [
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'myapp'
```



Django static files

- Define STATIC_URL in settings.py file

```
STATIC_URL = '/static/'
STATIC_ROOT= os.path.join(BASE_DIR,'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

- Load static files in the templates :

```
{% load static %}
```

Introduction to HTML

- **HTML stands for Hyper Text Markup Language**
- **What**: describes the structure of Web pages using markup, HTML elements are the building blocks of HTML pages, HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on Browsers do not display the HTML tags, but use them to render the content of the

HTML Tag

<html> Defines an HTML document

<head> Defines information about the document

<title> Defines a title for the document

<body> Defines the document's body

<h1> to <h6> Defines HTML headings

<p> Defines a paragraph

**
**Inserts a single line break

<hr> Defines a thematic change in the content

<!--...--> Defines a comment

HTML Table Tag

TR:- Use for table Row

TD:- Use for Table data

Tbody:--Create table inside table

Caption:- Use for table title

TH:--Use for column Title

Thead:- Table header

Tfoot:- Table footer

Attributes:

Cellpadding:- space between text and cell border

Cellspacing:- space between cell and table border

Colspan/Rowspan :-- Merge columns

bgcolor:- Background color

border:-- define table border

TOPS Technologies

HTML Forms

- Types of input we can collect via forms
- Text via text boxes and text areas
- Selections via radio buttons, check boxes, pull down menus, and select boxes
- Form actions do all of the work
- Usually through button clicks

HTML5

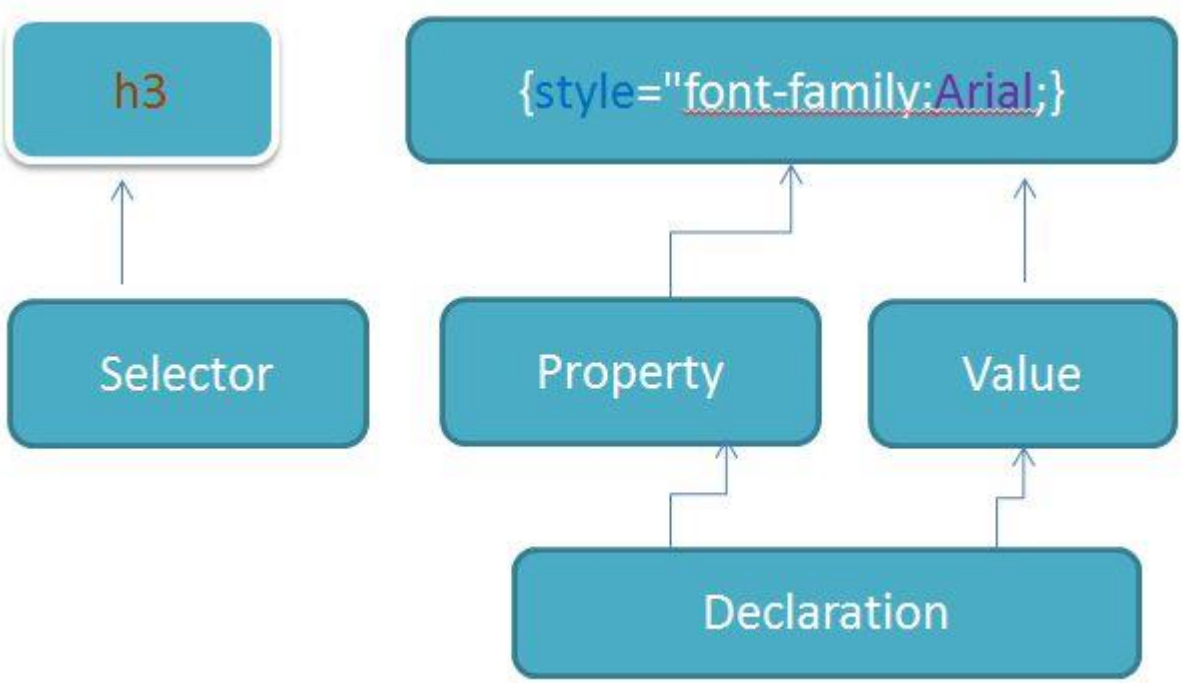
- **Form Input Tags:**
  Color,date,datetime,datetime-local,email,month,number,range,search, time,url,week
- **Form Input Attributes:**
  Autocomplete,autofocus,form,formaction,formenctype,formmethod,formnovalidate,formtarget,height and width,list,min and max,multiple, pattern (regexp),placeholder,required,step
- **Media Tags:-**Audio,video
- **Helpers(Plug-ins):**Object,Embed,Youtube video

Introduction to CSS

- **What:** Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.
  CSS handles the look and feel part of a web page
- **Why:** CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.
- **Who:** The CSS Working Group creates documents called specifications. When a specification has been discussed and officially ratified by the W3C members, it becomes a recommendation.

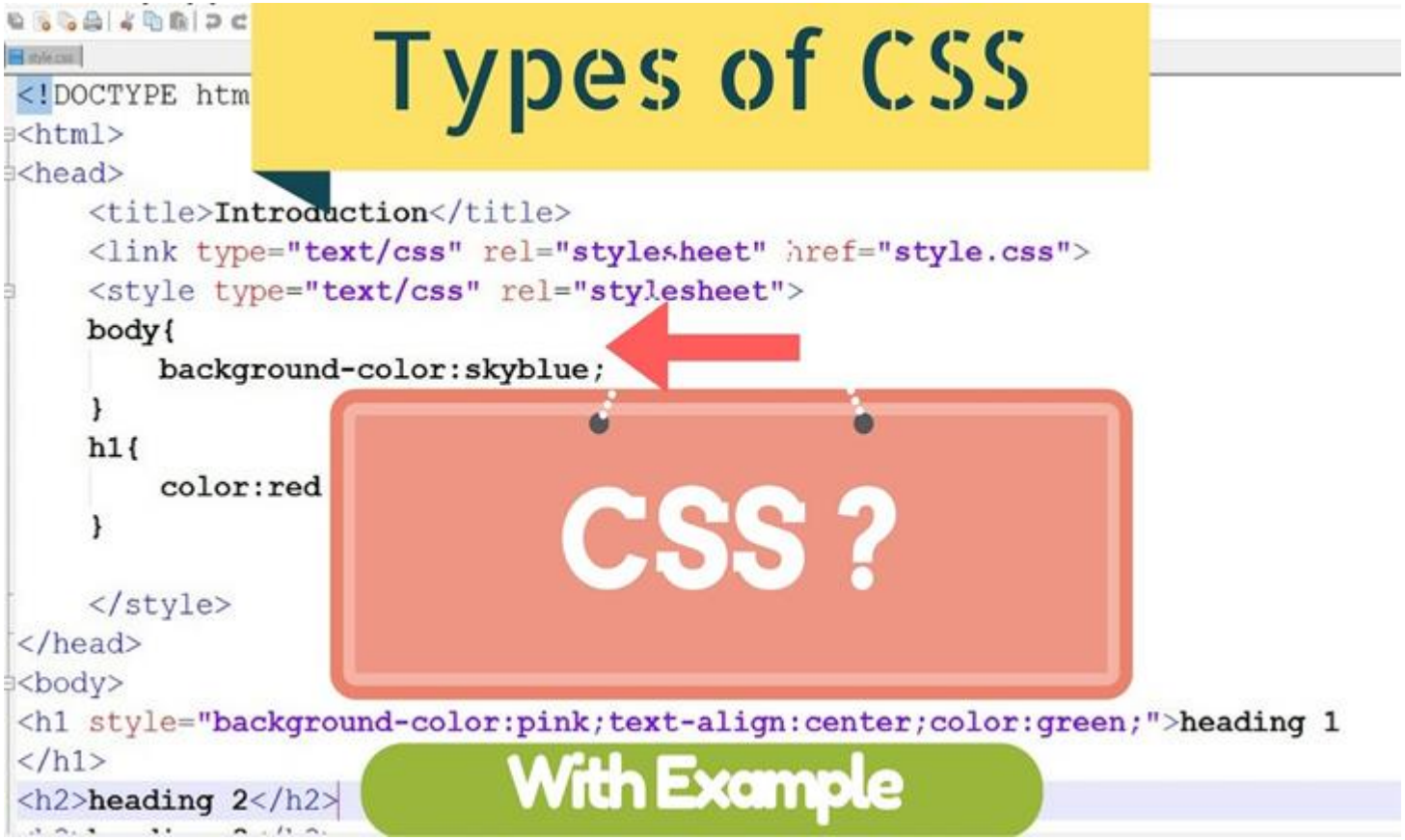
CSS Selector

- SELECTOR { PROPERTY: VALUE }
- HTML tag"{ "CSS Property": "Value" ; }



CSS Types

- Internal
- External
- Inline



TOPS Technologies

Margin and Padding

- Internal **Margin** and **padding** are the two most commonly used properties for spacing-out elements. A margin is the space outside of the element, whereas padding is the space inside the element

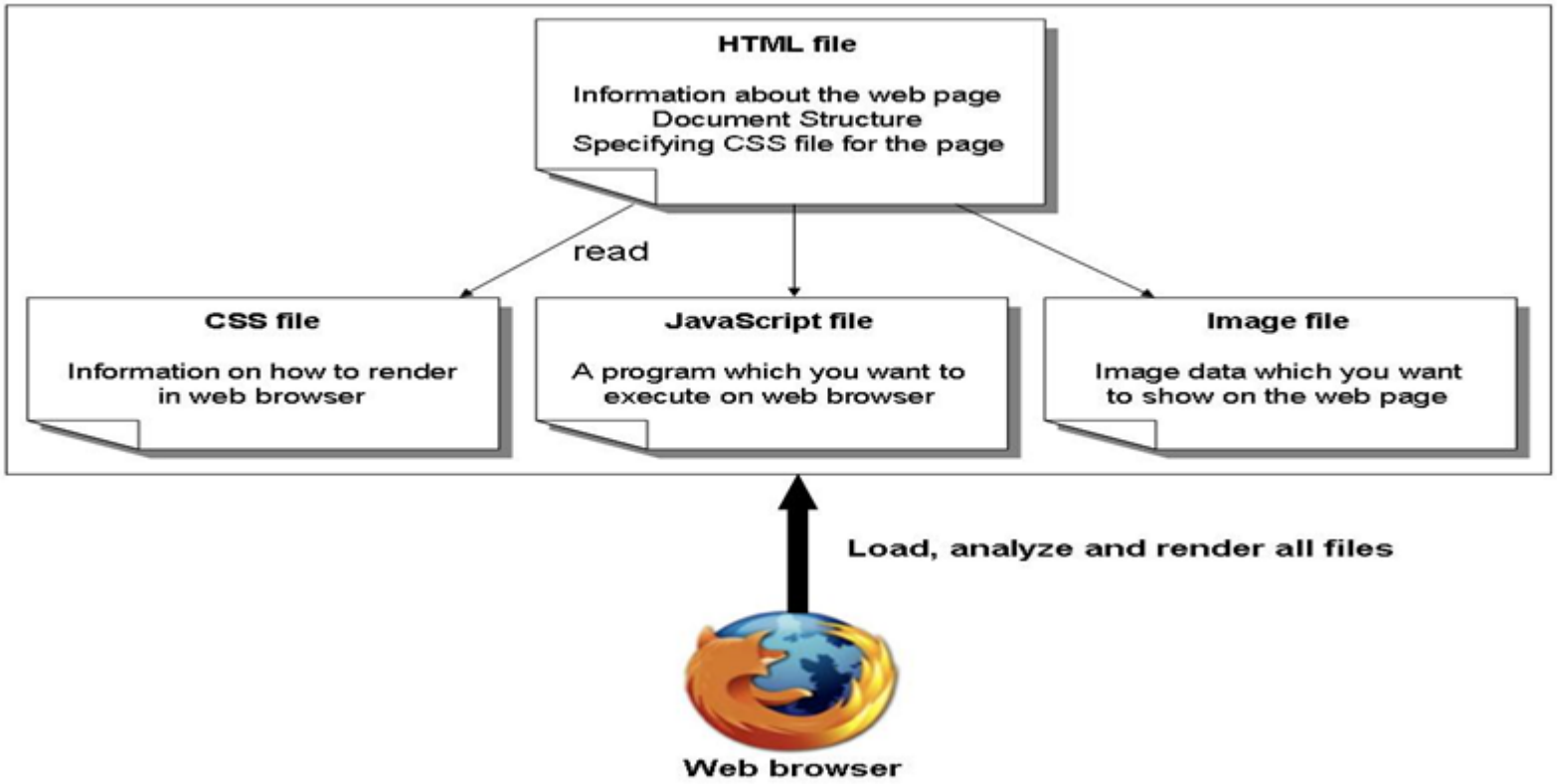
- The four sides of an element can also be set individually. **margin-top, margin-right, margin-bottom, margin-left, padding-top, padding-right, padding-bottom and padding-left** are the self-explanatory properties you can use.

CSS classes and ID

Class	ID
Class Can call by using '.' operator	ID can call by using # operator
Class can multiple in a single tag at a time	ID can only one in a tag at a time
Class has less priority then ID	Class has high priority then Class

Java Script

- **What: JavaScript** is an open source & most popular client side scripting language supported by all browsers. JavaScript is used mainly for enhancing the interaction of a user with the webpage



Events

- **What:** Javascript has events to provide a dynamic interface to a webpage. These events are hooked to elements in the Document Object Model(DOM).

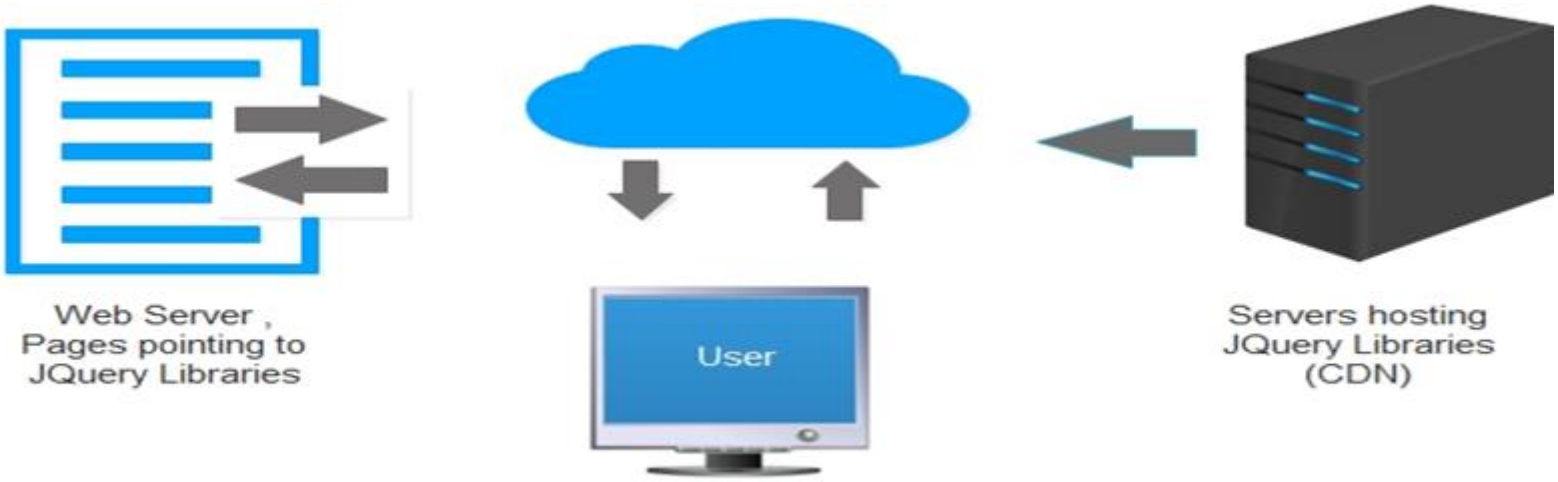
Event name	Event Source	Event handler
abort	image	onAbort
click	checkbox, radio button, submit button, reset button & link	onClick
change	text field/area, list	onChange
dragDrop	window	onDragDrop
error	image, window	onError
keyDown	doc, image, link	onKeyDown
keyPress	doc, image, link	onKeyPress
mouseMove	nothing	onMouseMove
mouseOut	link, image map	onMouseOut
mouseOver	link, image map	onMouseOver
reset	reset button form	onReset
resize	window	onResize
submit	submit button form	onSubmit

Jquery

- jQuery is the most popular JavaScript library nowadays. It uses CSS selector style syntax to fetch elements in document object model (DOM) into wrapped element set, and then manipulate elements in the wrapped set with jQuery functions to archive different effect.

- Though the way of using jQuery is very easy and intuitive, we should still understand what is happening behind the scene to better master this JavaScript library.

Jquery

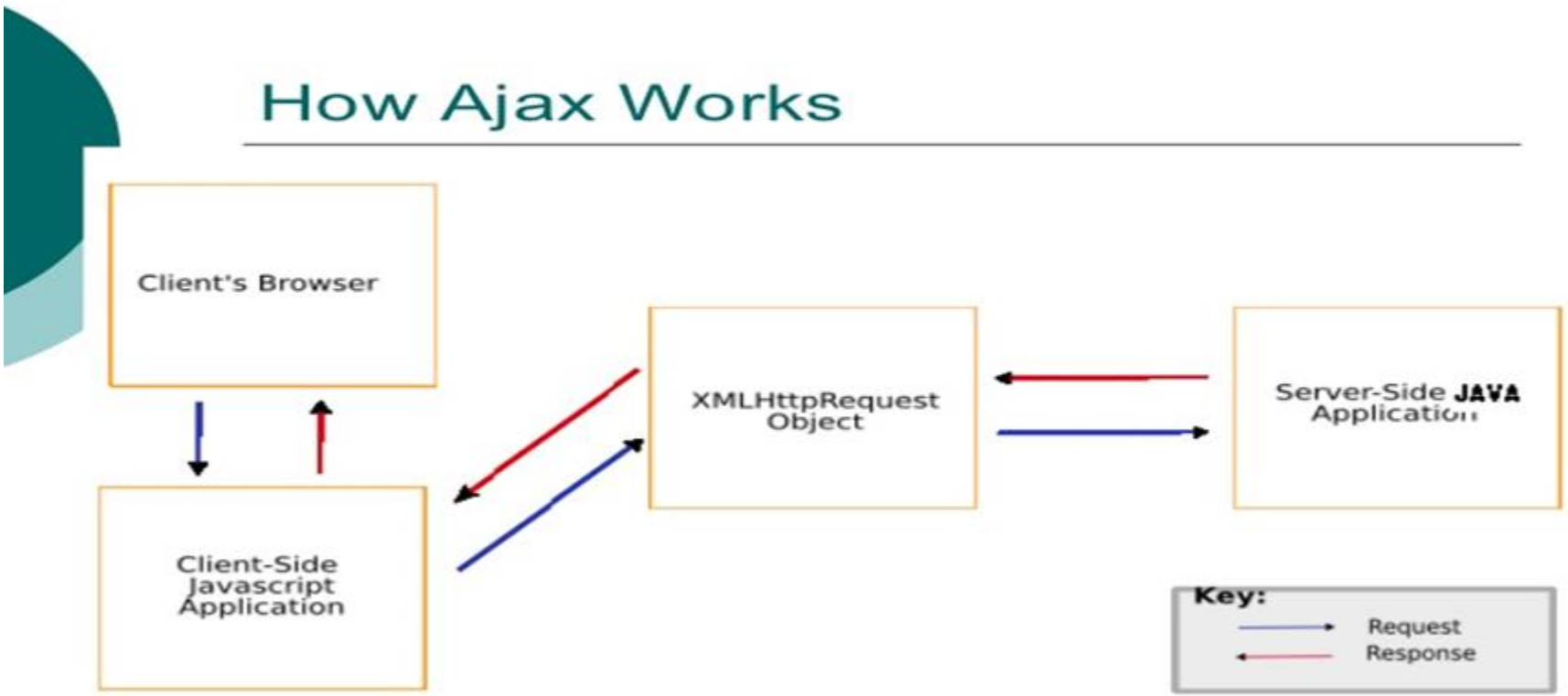


TOPS Technologies

Ajax (Asynchronous Javascript And XML)

- **What**: client-sided web development technique that is used to produce interactive Web applications. AJAX is a way of developing an application that combines the functions below, using JavaScript to tie it all together.

Ajax



TOPS Technologies

Django forms

- Django provides a Form class which is used to create HTML forms.
- It is similar to the ModelForm class that creates a form by using the Model, but it does not require the Model.
- Each field of the form class map to the HTML form <input> element and each one is a class itself, it manages form data and performs validation while submitting the form.

Django forms

- {{ form.as_table }} will render them as table cells wrapped in <tr> tags
- {{ form.as_p }} will render them wrapped in <p> tags
- {{ form.as_ul }} will render them wrapped in <li> tags

Django forms example

- https://github.com/TopsCode/Python/tree/master/Module-13/13.1%20Django%20Framework/django-form

TOPS Technologies

Django Model forms

- It is a class which is used to create an HTML form by using the Model. It is an efficient way to create a form without writing HTML code.
- Django automatically does it for us to reduce the application development time.
- For example, suppose we have a model containing various fields, we don't need to repeat the fields in the form file.

Django Model forms example

- https://github.com/TopsCode/Python/tree/master/Module-13/13.1%20Django%20Framework/django-model-forms/Django-Demos-feature-django-model-forms

Database Migrations

- Migration is a way of applying changes that we have made to a model, into the database schema.

- Django creates a migration file inside the migration folder for each model to create the table schema, and each table is mapped to the model of which migration is created.

Database Migrations

- **makemigrations :** It is used to create a migration file that contains code for the tabled schema of a model.
- **migrate :** It creates table according to the schema defined in the migration file.
- **showmigrations :** It lists out all the migrations and their status.

Django project (CRUD and E-Mail implementation)

- https://github.com/TopsCode/Python/tree/master/Module-13/13.2%20Django%20CRUD%20Operations/Trainer-Board-Project-master/Trainer-Board-Project-master/trainer_board

Thank You

TOPS Technologies