MODULE I

Basics of Software Development including coding standards, Python - Introduction to Python, tools for working with data in Python, Basic Syntax, Variables, expressions and Data Types, Working with Python: Numbers and String, Python Operators, Python General Programs, Input and output statements in python, reading data from keyboard, type conversions, String Manipulations - Accessing String, Basic Operations, String slices, Functions and Method, String formatting.

Software Development.

▪Software development is a process of writing and maintaining the source code.

▪In a broader sense, it includes all that is involved between the conception of the desired software through to the final manifestation of the software.

▪Software development may include research, new development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

▪Programming is making decisions, or telling the computer what decisions to make under different circumstances.

Programming and Coding

▪Programming is making decisions, or telling the computer what decisions to make under different circumstances.

Coding is a subset of programming which encompasses the following

- Programming languages
- A language's syntax and how it differs from other language's syntax
- Code arrangement
- Code optimization
- Debugging
- Writing and running tests
- Creating and using libraries and frameworks

Coding Standards.

▪Think of coding standards as a set of rules, techniques, and best practices to create cleaner, more readable, more efficient code with minimal errors.

▪They offer a uniform format by which software engineers can use to build sophisticated and highly functional code.

Advantages of implementing Coding Standards

- Offers uniformity to the code created by different engineers.
- Enables the creation of reusable code.
- Makes it easier to detect errors.
- Make code simpler, more readable, and easier to maintain.
- Boost programmer efficiency and generates faster results.

Choose industry-specific coding standards.

▪Adhering to industry-specific standards makes it easier to write accurate code that matches product expectations.

▪The standards required for coding software for luxury automobiles will differ from those for coding software for gaming.

▪It becomes easier to write code that will satisfy the end-users and meet business requirements.

Focus on code readability : Readable code is easy to follow, optimizes space and time. Here are a few ways to achieve that:

▪Write as few lines as possible.
▪Use appropriate naming conventions.
▪Segment blocks of code in the same section into paragraphs.
▪Use indentation to marks the beginning and end of control structures.
▪Don't use lengthy functions. Ideally, a single function should carry out a single task.
▪Use the DRY (Don't Repeat Yourself) principle and Automate repetitive tasks
▪Avoid Deep Nesting as Too many nesting levels make code harder to read.
▪Avoid long lines as It is easier for humans to read blocks of lines that are horizontally short and vertically long.

Basics of Software Development Including Coding Standards

▪**Standardize headers for different modules** : It is easier to understand and maintain code when the headers of different modules align with a singular format. For example, each header should contain:

▪Module Name
▪Date of creation
▪Name of creator of module
▪History of modification
▪Summary of what the module does
▪Functions in that module
▪Variables accessed by the module

Basics of Software Development Including Coding Standards

Don't use a single identifier for multiple purposes

Turn daily backups into an instinct

Leave comments and prioritize documentation

Try to formalize Exception Handling

▪Keep the code in a try-catch block.
▪Ensure that auto recovery has been activated and can be used.
▪Consider that it might be an issue of software/network slowness. Wait a few seconds for the required elements to show up.
▪Use real-time log analysis.

Introduction to Python

❖ Python was created by G.V. Rossum during 1985- 1990 and released in 1991.

❖Technical Python is a widely used high-level programming language for general purpose programming

❖Python features a dynamic type system and automatic memory management.

❖Python supports multiple programming paradigms, including object-oriented, imperative, functional programming and procedural styles.

❖Python has a large and comprehensive standard library..

Characteristics of Python

❖ Python supports functional and structured programming methods as well as OOP.

❖ Python can be used as a scripting language or can be compiled to byte-code for building large applications.

❖Python provides high-level dynamic data types and dynamic type checking.

❖Python supports automatic garbage collection.

❖Python can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Features of Python

❖ **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax.

❖**Easy-to-read** – Python code is more clearly defined and visible to the eyes.

❖ **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.

❖**A broad standard library** – Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

❖**Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Applications of Python

❖ **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

❖ **Extendable** – You can add low-level modules to the Python interpreter which enable programmers to add to or customize their tools to be more efficient.

❖ **Databases** – Python provides interfaces to all major commercial databases.

❖ **GUI Programming** – Python supports GUI applications

❖**Scalable** – Python provides a better structure and support for large programs than shell scripting

Python Uses

- ❖Machine Learning
- ❖GUI Applications (like Kivy, Tkinter, PyQt etc.)
- ❖Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- ❖Image processing (like OpenCV, Pillow)
- ❖Web scraping (like Scrapy, BeautifulSoup, Selenium)
- ❖Test frameworks , Multimedia, Scientific computing, and Text processing

RA20 APTT

Development Tools

- ❖ A developer uses many tools like editors, libraries, compiling and testing platforms.
- ❖IDE helps to automate the task of a developer by reducing manual efforts.
- ❖IDE is a software pack which are used for developing and testing the software.
- ❖ If IDE is not present, then the developer has to manually do the selections, integrations and deployment process.
- ❖ IDE is basically developed to simplify the SDLC process, by reducing coding and avoiding typing errors.

RA20 APTT

What is IDE and Text or Code Editor?

IDE: IDE is a development environment which provides many features like coding, compiling, debugging, executing, auto complete, libraries, in one place for the developer's thus making tasks simpler

TEXT OR CODE EDITOR

- ❖Text editor helps the programmer for writing scripts, modifying code or text etc.
- ❖Text editor allows modifying all types of files rather than specifying any particular language or types

RA20 APTT

Top Python IDEs and Code Editors

IDE	User Rating	Size in MB	Developed in
PyCharm	4.5/5	BIG	JAVA, PYTHON
Spyder	May 4, 2018	BIG	PYTHON
PyDev	4.6/5	MEDIUM	JAVA, PYTHON
Idle	4.2/5	MEDIUM	PYTHON
Wing	May 4, 2018	BIG	C, C++, PYTHON

RA20 APTT

- This IDE created by JetBrains.
- It has an incredible reputation within the Python developer community.

Some of its key features are:

-Integration with frameworks such as

-Django, Flask, Pyramid or Web2Py.

- Auto-completion.

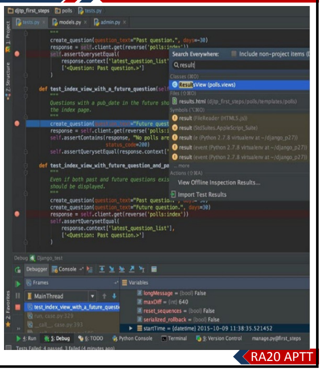
- Syntax highlighter.

- Analysis tool.

-Refactoring.

-Advanced Python

-JavaScript debugger.

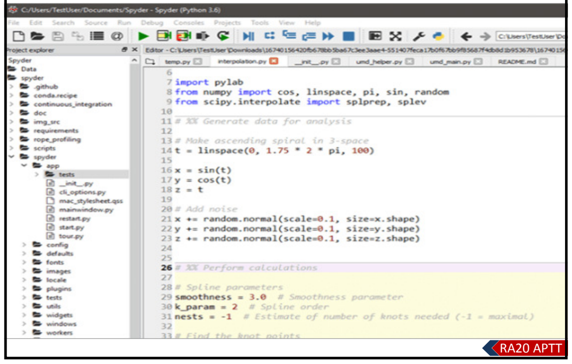

djtp_first_steps
polls
tests.py
models.py
admin.py
1: Project
Z: Structure
Search Everywhere:
Include non-project items
result
Classes
ResultsView (polls.views)
Files
results.html (djtp_first_steps/polls/templates/polls)
Symbols
result (FileReader (HTML5.js))
result (StdSuites.AppleScript_Suite)
result (e (Python 2.7.8 virtualenv at ~/django_p27))
result (event (Python 2.7.8 virtualenv at ~/django_p27))
result (event (Python 2.7.8 virtualenv at ~/django_p27))
... more
Actions
View Offline Inspection Results...
Import Test Results
Debug Django_test
Debugger
Console
Frames
Variables
MainThread
test_index_view_with_a_future_questi
run, case.py:329
__call__, case.py:393
2: Favorites
4: Run
5: Debug
6: TODO
Python Console
Terminal
9: Version Control
manage.py@first_steps
Tests Failed: 4 passed, 3 failed (4 minutes ago)

```
        create_question(question_text="Past question.", days=-30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'
            ['<Question: Past question.>']
        )

    def test_index_view_with_a_future_question(self
        """
        Questions with a pub_date in the future sho
        the index page.
        """
        create_question(question_text="Future quest
        response = self.client.get(reverse('polls:i
        self.assertContains(response, "No polls are
                            status_code=200)
        self.assertQuerysetEqual(response.context['

    def test_index_view_with_future_question_and_pa
        """
        Even if both past and future questions exis
        should be displayed.
        """
        create_question(question_text="Past questio
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question.>']
        )
```

```
longMessage = {bool} False
maxDiff = {int} 640
reset_sequences = {bool} False
serialized_rollback = {bool} False
startTime = {datetime} 2015-10-09 11:38:35.521452
```

RA20 APTT

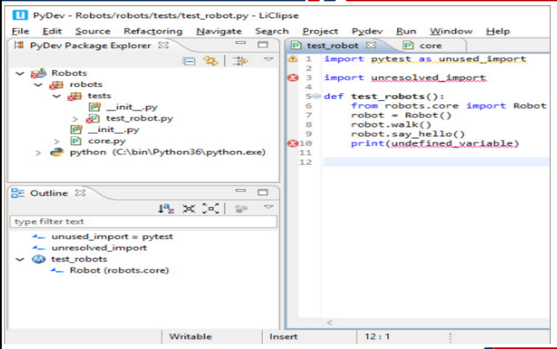
SPYDER



```

import pylab
from numpy import cos, linspace, pi, sin, random
from scipy.interpolate import splprep, splev

# %% Generate data for analysis

# Make ascending spiral in 3-space
t = linspace(0, 1.75 * 2 * pi, 100)

x = sin(t)
y = cos(t)
z = t

# Add noise
x += random.normal(scale=0.1, size=x.shape)
y += random.normal(scale=0.1, size=y.shape)
z += random.normal(scale=0.1, size=z.shape)


# %% Perform calculations

# Spline parameters
smoothness = 3.0  # Smoothness parameter
k_param = 2  # Spline order
nests = -1  # Estimate of number of knots needed (-1 = maximal)

# Find the knot points
```

RA20 APTT

PyDev



PyDev - Robots/robots/tests/test_robot.py - LiClipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

PyDev Package Explorer

```
Robots
  robots
    tests
      __init__.py
      test_robot.py
    __init__.py
    core.py
  python (C:\bin\Python36\python.exe)
```

test_robot | core

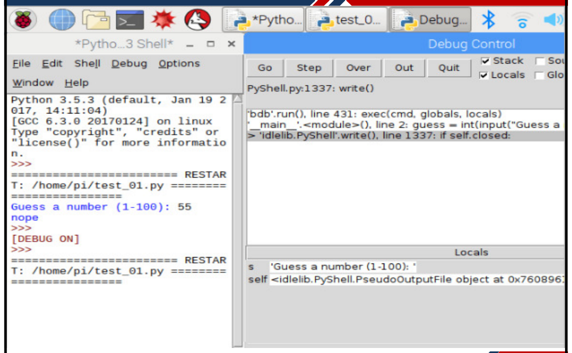
```
import pytest as unused_import

import unresolved_import

def test_robots():
    from robots.core import Robot
    robot = Robot()
    robot.walk()
    robot.say_hello()
    print(undefined_variable)


```

Outline

```
unused_import = pytest
unresolved_import
test_robots
  Robot (robots.core)
```

Writable | Insert | 12 : 1

Idle



Pytho...3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.3 (default, Jan 19 2
017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or
"license()" for more informatio
n.
>>>
======================== RESTAR
T: /home/pi/test_01.py ========
================
Guess a number (1-100): 55
nope
>>>
[DEBUG ON]
>>>
======================== RESTAR
T: /home/pi/test_01.py ========
================
```

Debug Control

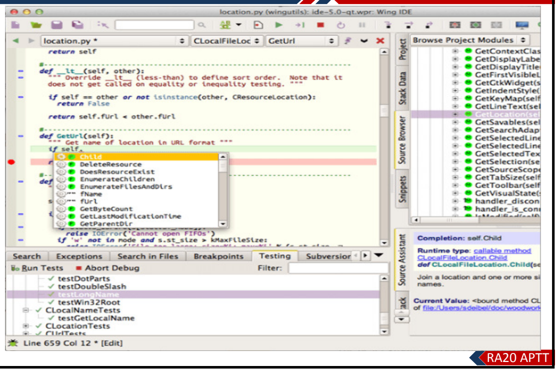
Go | Step | Over | Out | Quit | ✓ Stack | Sou | ✓ Locals | Glo

```
PyShell.py:1337: write()

'bdb'.run(), line 431: exec(cmd, globals, locals)
'__main__'.<module>(), line 2: guess = int(input("Guess a
> 'idlelib.PyShell'.write(), line 1337: if self.closed:
```

Locals

```
s     'Guess a number (1-100): '
self  <idlelib.PyShell.PseudoOutputFile object at 0x760896
```

Wing



location.py (wingutils): ide-5.0-qt.wpr: Wing IDE

location.py * | CLocalFileLoc | GetUrl

```
    return self

  #---------------------------------------------------------------
  def __lt__(self, other):
    """ Override __lt__ (less-than) to define sort order.  Note that it
    does not get called on equality or inequality testing. """

    if self == other or not isinstance(other, CResourceLocation):
      return False

    return self.fUrl < other.fUrl

  #---------------------------------------------------------------
  def GetUrl(self):
    """ Get name of location in URL format """
    if self.
```

```
Child
DeleteResource
DoesResourceExist
EnumerateChildren
EnumerateFilesAndDirs
fName
fUrl
GetByteCount
GetLastModificationTime
GetParentDir
```

```
      raise IOError('Cannot open FIFOs')
    if 'w' not in mode and s.st_size > kMaxFileSize:
```

Search | Exceptions | Search in Files | Breakpoints | Testing | Subversion

Run Tests | Abort Debug | Filter:

```
✓ testDotParts
✓ testDoubleSlash
✓ testLongName
✓ testWin32Root
✓ CLocalNameTests
  ✓ testGetLocalName
✓ CLocationTests
✓ CUrlTests
```

Line 659 Col 12 * [Edit]

Browse Project Modules

```
GetContextClas
GetDisplayLabe
GetDisplayTitle
GetFirstVisibleL
GetGtkWidget(s
GetIndentStyle(
GetKeyMap(self
GetLineText(sel
GetLocation(sel
GetSavables(sel
GetSearchAdap
GetSelectedLine
GetSelectedLine
GetSelectedTex
GetSelection(se
GetSourceScope
GetTabSize(self
GetToolbar(self
GetVisualState(
handler_discon
handler_is_conn
```

Completion: self.Child

Runtime type: callable method CLocalFileLocation.Child
def CLocalFileLocation.Child(se

Join a location and one or more si names.

Current Value: <bound method CL of file:/Users/sdeibel/doc/woodwork

Python Environment

❖ Python files have extension .py

❖Before we start Python programming, we need to have an interpreter to interpret and run our programs.

❖ IDLE (Integrated Development Environment

print(" Hello Python") → Hello Python

❖Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

Python Syntax

❖ Python syntax can be executed by writing directly in the Command Line.

:>>> print("Hello, World!")

❖By creating a python file on the server, using the .py file extension, and running it in the Command Line:

C:\Users\Your Name>python myfile.py

Python Indentation

❖ Indentation refers to the spaces at the beginning of a code line.

❖Where in other programming languages the indentation in code is for readability only.

❖ the indentation in Python is very important.

❖ Python uses indentation to indicate a block of code.

```
if 3 > 2:
print("Five is greater than two!")      SYNTAX ERROR
```

❖

```
if 5 > 2:
    print("Five is greater than two!")  RUN  SUCCESSFULLY
```

Contd..

❖ Indentation refers to the spaces at the beginning of a code line.

❖
```
if 5 > 2:
    print("Five is greater than two!")
        print("Five is greater than two!")  Unexpected Indent
```

❖
```
if 5 > 2:
    print("Five is greater than two!")
    print("Five is greater than two!")
```

Python Comments-----

❖Comments are lines that exist in computer programs that are ignored by compilers and interpreters.

❖Comments can be used to explain Python code.

❖Comments can be used to make the code more readable.

❖Comments can be used to prevent execution when testing code.

❖Comments starts with a # and Python will ignore them (#This is a comment).

❖" " " This is a comment written in more than just one line " " "

print("Hello, Python!")

Values and Variables

❖ A variable means reserving memory location to store values.

❖ In variable, you can store any kind of values by using appropriate data types.

❖In Python, variables do not need a declaration to reserve memory space.

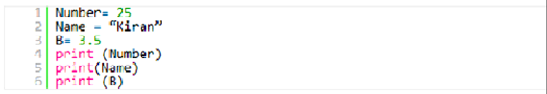
❖The "variable declaration" or "variable initialization" happens automatically when we assign a value to a variable.

❖Data types are nothing but variables you use to reserve some space in memory.

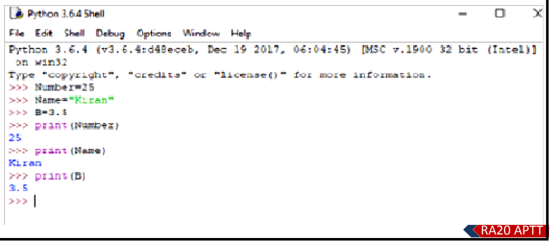
❖Python has no command for declaring a variable.

Contd..

How to Declare and use a Variable



```
Number= 25
Name = "Kiran"
B= 3.5
print (Number)
print(Name)
print (B)
```



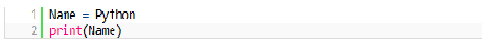
```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Number=25
>>> Name="Kiran"
>>> B=3.5
>>> print(Number)
25
>>> print(Name)
Kiran
>>> print(B)
3.5
>>> |
```

Contd..

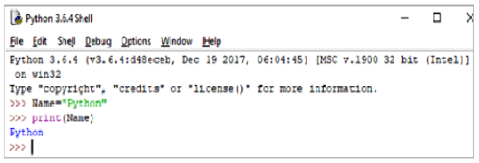
Re-declare a Variable

❖ We can re-declare a variable at any point of time even after you have declared it

Example:



```
Name = Python
print(Name)
```

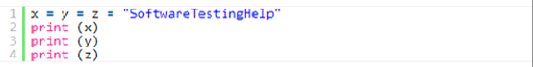


```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Name="Python"
>>> print(Name)
Python
>>> |
```

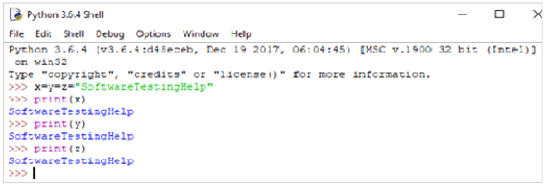
Contd..

Multiple Assignment

❖ In Python, we can assign the same value to multiple variables at once.



```
x = y = z = "SoftwareTestingHelp"
print (x)
print (y)
print (z)
```



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=y=z="SoftwareTestingHelp"
>>> print(x)
SoftwareTestingHelp
>>> print(y)
SoftwareTestingHelp
>>> print(z)
SoftwareTestingHelp
>>> |
```

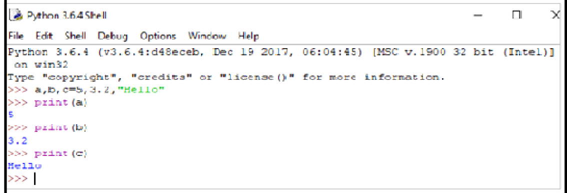
Contd..

SR UNIVERSITY

Multiple Assignment

❖ In Python, We can also assign multiple values to multiple variables.

```
a, b, c = 5, 3.2, "Hello"
print (a)
print (b)
print (c)
```



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> a,b,c=5,3.2,"Hello"
>>> print(a)
5
>>> print(b)
3.2
>>> print(c)
Hello
>>> |
```

Contd..

SR UNIVERSITY

❖ A variable name must start with a letter or the underscore character.

❖A variable name cannot start with a number.

❖A variable name can only have alpha-numeric characters (A-z, 0-9, and _).

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

Python Data Types

SR UNIVERSITY

❖ A Data Type describes the characteristic of a variable.

Python has six standard Data Types:

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

Numbers

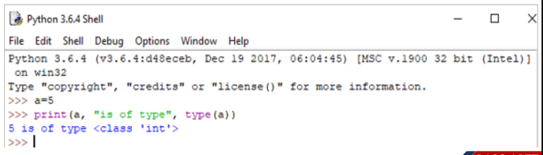
SR UNIVERSITY

❖ In Numbers, there are mainly 3 types which include Integer, Float, and Complex.

❖ These 3 are defined as a class in python. In order to find to which class the variable belongs to you can use type () function.

```
a = 5
print(a, "is of type", type(a))
```

Output: 5 is of type <class 'int'>



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=5
>>> print(a, "is of type", type(a))
5 is of type <class 'int'>
>>> |
```

Contd..

SR UNIVERSITY

int: Integer number
a = 2
b = 234567890
c =385638463264243 24

Integers in Python are of arbitrary sizes.

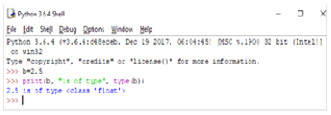
Float: In Python corresponds to a C double.

b = 100.e0
c = 123456789.e1

Complex:
a = 2 + 1j
b = 100 + 10j

```
b = 2.5
print(b, "is of type", type(b))
```

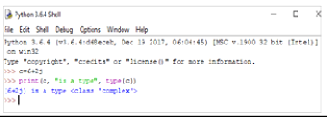
Output: 2.5 is of type <class 'float'>



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> b=2.5
>>> print(b, "is of type", type(b))
2.5 is of type <class 'float'>
>>> |
```

```
c = 6+2j
print(c, "is a type", type(c))
```

Output: (6+2j) is a type <class 'complex'>



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> c=6+2j
>>> print(c, "is a type", type(c))
(6+2j) is a type <class 'complex'>
>>> |
```

String

SR UNIVERSITY

❖ A string is an ordered sequence of characters.

❖ We can use single quotes or double quotes to represent strings ' ' or " ".

❖ Multi-line strings can be represented using triple quotes, ''' or " " ".

❖ Strings are immutable which means once we declare a string we can't update the already declared string.

```
Single = 'Welcome'
or
Multi = "Welcome"
```

String

SR UNIVERSITY

```
# defining strings in Python
# all of the following are equivalent
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
           the world of Python"""
print(my_string)
```

Output

```
Hello
Hello
Hello
Hello, welcome to
           the world of Python
```

```
#Accessing string characters in Python
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

Output

```
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
```

RA20 APTT

String

SR UNIVERSITY

Strings are immutable : This means that elements of a string cannot be changed once they have been assigned.

```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the **del** keyword.

```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

RA20 APTT

Python String Operations

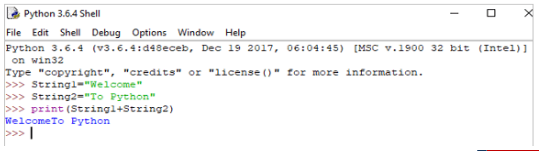
SR UNIVERSITY

- There are many operations that can be performed with strings which makes it one of the most used data types in Python.

Concatenation: It means the operation of joining two strings together.

```
String1 = "Welcome"
String2 ="To Python"
print(String1+String2)
```

Output: Welcome To Python



Python 3.6.4 Shell

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> String1="Welcome"
>>> String2="To Python"
>>> print(String1+String2)
WelcomeTo Python
>>> 
```

RA20 APTT

Contd..

SR UNIVERSITY

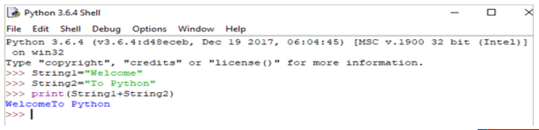
Multiline:

- "'Python is an interpreted high-level programming language'"

Concatenation: It means the operation of joining two strings together.

```
String1 = "Welcome"
String2 ="To Python"
print(String1+String2)
```

Output: Welcome To Python



Python 3.6.4 Shell

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> String1="Welcome"
>>> String2="To Python"
>>> print(String1+String2)
WelcomeTo Python
>>> 
```

RA20 APTT

Contd..

SR UNIVERSITY

- The + operator concatenates two string literals together.
- The * operator can be used to repeat the string for a given number of times..
- Writing two string literals together also concatenates them like + operator. If we want to concatenate strings in different lines, we can use parentheses.

```
# Python String Operations
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)


str1 + str2 =  HelloWorld!
str1 * 3 = HelloHelloHello
```

```
>>> # two string literals together
>>> 'Hello ''World!'
'Hello World!'

>>> # using parentheses
>>> s = ('Hello '
...      'World')
>>> s
'Hello World'
```

RA20 APTT

Iterating Through a string

SR UNIVERSITY

- We can iterate through a string using a for Loop. Here is an example to count the number of 'l's in a string..

```
# Iterating through a string
count = 0
for letter in 'Hello World':
    if(letter == 'l'):
        count += 1
print(count,'letters found')


3 letters found
```

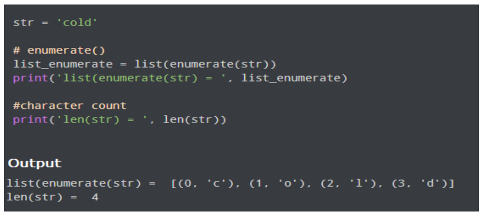
String Membership Test: We can test if a substring exists within a string or not, using the keyword in.

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

RA20 APTT

Built-in functions for Strings

- The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs.
- Similarly, len() returns the length (number of characters) of the string.



```
str = 'cold'

# enumerate()
list_enumerate = list(enumerate(str))
print('list(enumerate(str) = ', list_enumerate)

#character count
print('len(str) = ', len(str))
```

Output

```
list(enumerate(str) =  [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
len(str) =  4
```

RA20 APTT

Accessing

word = "Hello World"
>>> len(word)
11

>>> print word. count('l')
3

s = "Count, the number of spaces"

>>> print s.count(' ')

28

word[start:end]

word[start:]

word[:end]

word[:]

print word[0] - one char of the word

print word[0:1] -one char of the word

print word[0:3] -get the first three char

print word[:3] -get the first three char

print word[-3:] -get the last three char

print word[3:] -all Not first three char

print word[:-3] -All but three last char

Output

H   , H   Hel   Hel

rld

lo World

Hello Wo

RA20 APTT

Split Strings

word = **"Hello World"**

# Split on whitespace

>>> word. split(' ')   **['Hello', 'World']**

>>> word.startswith("H")   **True**

>>> word.endswith("d")   **True**

>>> word.endswith("w")   **False**

Repeat Strings

Print "."* 10
>>> print “,” * 10   ,,,,,,,,,,

>>> word.replace("Hello", "Goodbye")   **`Goodbye World**

RA20 APTT

word = **"Hello World"**

>>> print string.upper()
HELLO WORLD

>>> print string.lower()
hello world

>>> print string.title()
Hello World

>>> print string.capitalize()
Hello world

>>> print string.swapcase()
hELLO wORLD

Reversing A String

string = "Hello World“

>>> print ' '.join(reversed(string))

d l r o W o l l e H

Strip

Python strings have the strip(), lstrip(), rstrip() methods for removing any character from both ends of a string.

>>> **print word.strip()**   **xyz**

>>> print **word.lstrip()**   **xyz**

>>> **print word.rstrip()**   **xyz**

RA20 APTT

Join

>>> print “ : ".join(word)   **# add a : between every char H:e:l:l:o: :W:o:r:l:d**

>>> print " ".join(word)   **# add whitespace between every char H e l l o W o r l d**

Testing : A string in Python can be tested for truth value.

The return type will be in Boolean value (True or False)

1. **word.isalnum()**   **#check if all char are alphanumeric**
2. word.isalpha()   #check if all char in the string are alphabetic
3. **word.isdigit()**   **#test if string contains digits**
4. word.istitle()   #test if string contains title words
5. **word.isupper()**   **#test if string contains upper case**
6. word.islower()   #test if string contains lower case
7. **word.isspace()**   **#test if string contains spaces**
8. word.endswith('d')   #test if string endswith a d
9. **word.startswith('H')**   **#test if string startswith H**

RA20 APTT

How to change or delete a string?

We cannot delete or remove characters from a string.

But deleting the string entirely is possible using the del keyword.

>>> my_string = ‘Hello >>>

my_string[3] = ‘L’ ... **TypeError: 'str' object does not support item assignment**

>>> del my_string

>>> my_string

...

NameError: name 'my_string' is not defined

RA20 APTT

String Slicing

•The slice() method returns a portion of an iterable as an object of the slice class based on the specified range. It can be used with string, list, tuple, set,

Syntax: **slice (stop)** **slice(start, stop, step)**

Parameters:

start: Starting index where the slicing of the iterable starts.
stop: Ending index where the slicing should end.
step: (Optional) An integer to increment starting index..

Return Value: Returns an object of the slice class.

String Slicing Examples

```
String  =  "Example of Slice Function in Python Language“
String1  =  "  Welcome To Python Programming";
print("The Length of the String variable is = ", len(String1) )

s1 = slice(3)
print("s1 = ",  String[s1] )  #  Print first 3 char     Exa

s2 = slice(1,5)
print("s2 = ", String[s2] )  #  Print first 1-4 char     xamp

s3 = slice(0,10,2)
print("s3 = ", String[s3]) #  Print alternate char from 0 to <10     Eapeo

print("Replace Function Used in String1 ",

String1.replace("Programming","Language"))
```

List

❖ A list can contain a series of values. and are declared by using bracket [ ].

❖ A list is mutable, which means we can modify the list.

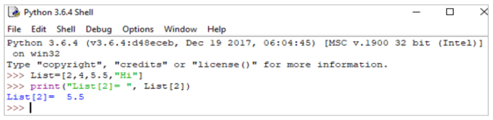
❖ lists are almost similar to arrays in C.

❖ One difference is that all the items belonging to a list can be of different data type.

list = [123,'abcd',10.2,'d']

```
List = [2,4,5.5,"Hi"]
print("List[2] = ", List[2])
```

Output: List[2] = 5.5



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> List=[2,4,5.5,"Hi"]
>>> print("List[2]= ", List[2])
List[2]=  5.5
>>> 
```

Accept List as an Input from a user

❖ Like string and integer, we can also accept list and array as an input from a user. For example, accept a list of numbers from the user.

Python List Input

1, 2, 3, 4, 5, 6, 7, 8

Red, Green, Yellow

Tuple

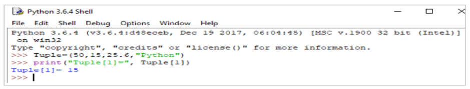
❖ A tuple is a sequence of Python objects separated by commas.

❖ Tuple are immutable, which means tuple once created cannot be modified.

❖ Tuples are defined using parentheses ().

```
Tuple = (50,15,25.6,"Python")
print("Tuple[1] = ", Tuple[1])
```

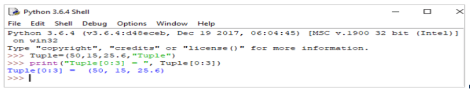
Output: Tuple[1] = 15



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Tuple=(50,15,25.6,"Python")
>>> print("Tuple[1]=", Tuple[1])
Tuple[1]= 15
>>> 
```

```
print("Tuple[0:3] =", Tuple[0:3])
```

Output: Tuple[0:3] = (50, 15, 25.6)



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Tuple=(50,15,25.6,"Tuple")
>>> print("Tuple[0:3] = ", Tuple[0:3])
Tuple[0:3] =  (50, 15, 25.6)
>>> 
```

Set

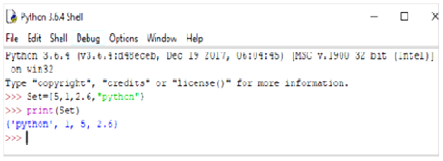
❖ A set is an unordered collection of items.

❖ Set is defined by values separated by a comma inside braces { }.

❖ In the set, we can perform operations like union and intersection on two sets.

```
Set = {5,1,2.6,"python"}
print(Set)
```

Output: {'python', 1, 5, 2.6}

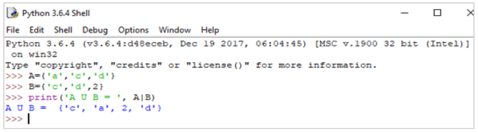


```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Set={5,1,2.6,"python"}
>>> print(Set)
{'python', 1, 5, 2.6}
>>> 
```

Contd..

```
A = {'a', 'c', 'd'}
B = {'c', 'd', 2 }
print('A U B =', A| B)
```

Output: A U B = {'c', 'a', 2, 'd'}



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> A={'a','c','d'}
>>> B={'c','d',2}
>>> print('A U B = ', A|B)
A U B =  {'c', 'a', 2, 'd'}
>>> 
```

We can perform Intersection operation by Using & Operator.

```
A = {100, 7, 8}
B = {200, 4, 7}
print(A & B)
```

Output: {7}

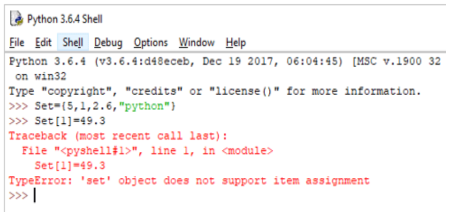
RA20 APTT

Contd..

- As the set is an unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

```
Set[1] = 49.3
```

Output: TypeError: 'set' object does not support item assignment



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Set={5,1,2.6,"python"}
>>> Set[1]=49.3
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    Set[1]=49.3
TypeError: 'set' object does not support item assignment
>>> 
```

RA20 APTT

Dictionary

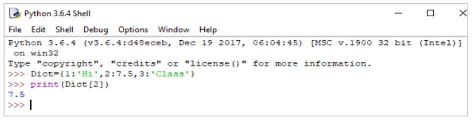
- Dictionaries are the most flexible built-in data type in python.
- Dictionaries items are stored and fetched by using the key.
- Dictionaries are used to store a huge amount of data.
- To retrieve the value we must know the key.
- In Python, dictionaries are defined within braces { }.
- We use the key to retrieve the respective value, but not the other way around.

RA20 APTT

Contd..

```
print(Dict[2])
```

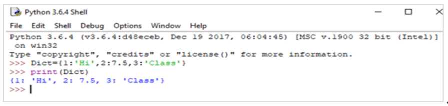
Output: 7.5



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> print(Dict[2])
7.5
>>> 
```

```
Dict = {1:'Hi',2:7.5, 3:'Class'}
print(Dict)
```

Output: {1: 'Hi', 2: 7.5, 3: 'Class'}



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> print(Dict)
{1: 'Hi', 2: 7.5, 3: 'Class'}
>>> 
```

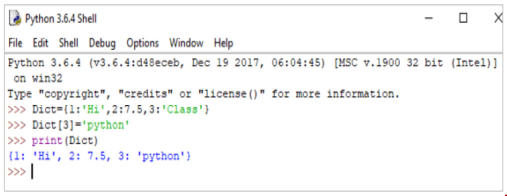
RA20 APTT

Contd..

```
Dict[3] = 'python'
print(Dict)
```

Output:

{1: 'Hi', 2: 7.5, 3: 'python'}



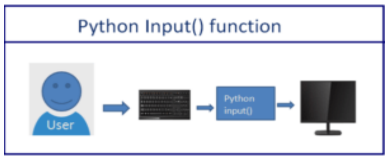
```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> Dict[3]='python'
>>> print(Dict)
{1: 'Hi', 2: 7.5, 3: 'python'}
>>> 
```

RA20 APTT

Input and Output Functions

input( ) function:

- Developers often have a need to interact with users to get data.
- Python provides us with two inbuilt functions to read the input from the keyboard.



RA20 APTT

Contd..

❖ There are different types of Input, and that comes in various ways.

❖Input stems from the keyboard. i.e., the user entered some value using a keyboard.

❖Input using Mouse Click or movement, i.e. you clicked on the radio button or some drop-down list and chosen an option from it.

❖User Input from the command line environment or through the user interface.

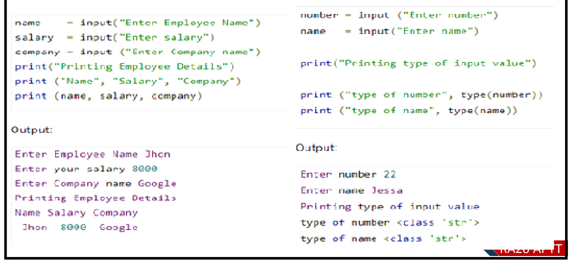
❖In both cases, the user is sending input from Keyboard or mouse.

Contd..

❖ When input() function executes, program flow stops until user enters some value.

❖Whatever you enter as input, input() function convert it into a string.

❖ An explicit conversion into an integer in your code is needed



```
name    = input("Enter Employee Name")
salary  = input("Enter salary")
company = input ("Enter Company name")
print("Printing Employee Details")
print ("Name", "Salary", "Company")
print (name, salary, company)
```

Output:

```
Enter Employee Name Jhon
Enter your salary 8000
Enter Company name Google
Printing Employee Details
Name Salary Company
 Jhon  8000  Google
```

```
number = input ("Enter number")
name   = input("Enter name")

print("Printing type of input value")

print ("type of number", type(number))
print ("type of name", type(name))
```

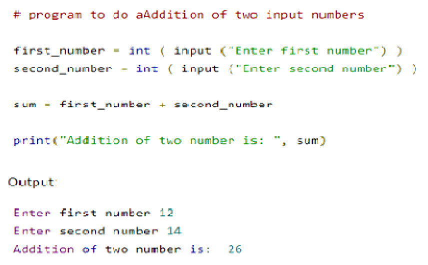
Output:

```
Enter number 22
Enter name Jessa
Printing type of input value
type of number <class 'str'>
type of name <class 'str'>
```

Contd..

Accept an Integer input from User

❖We need to convert an input value into an integer type explicitly.



```
# program to do aAddition of two input numbers

first_number = int ( input ("Enter first number") )
second_number = int ( input ("Enter second number") )

sum = first_number + second_number

print("Addition of two number is: ", sum)
```

Output

```
Enter first number 12
Enter second number 14
Addition of two number is:  26
```

Expressions and Arithmetic

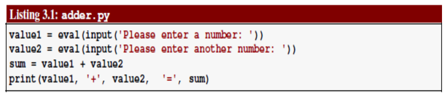
Expressions

A literal value like 34 and a variable like x are examples of a simple expressions. Values and variables can be combined with operators to form more complex expressions.

sum = value1 + value2;

•This is an assignment statement because is contains the assignment operator (=).

•To the right of the assignment operator is an arithmetic expression involving two variables and



Listing 3.1: adder.py

```
value1 = eval(input('Please enter a number: '))
value2 = eval(input('Please enter another number: '))
sum = value1 + value2
print(value1, '+', value2, '=', sum)
```

x = 10, y = 10

1. x + y
   If x and y are numbers
   • **10 + 10 = 20**
   If x and y are string
   • 10 + 10 = 1010
2. x - y
   If x and y are numbers
   • **10 - 10 = 0**
3. x * y
   If x and y are numbers
   • **10 * 10 = 100**
   If x is string and y is number
   • 10101010101010101010
   If x is number and y is string
   • 10101010101010101010

4. x / y
   If x and y are numbers
   10 /10 = 1
5. x // y
   If x and y are numbers
   11 // 10 = 1 floor Div
6. x % y
   If x and y are numbers
   10 % 10 = 0
7. x ** y
   If x and y are numbers
   10 ** 10 = 10 10 = 10000000000

Python Operators

❖Operators are used to perform operations on variables and values.

❖Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic Operators

Operator	Meaning	Example
+	Add two operands or unary plus	**x + y** +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

RA20 APTT

Arithmetic Operations

```
x = 15
y = 10
print('x + y =', x+y)
```

Output: x + y = 25

```
print('x - y =', x-y)
```

Output: x – y = 5

```
print('x * y =', x*y)
```

Output: x * y = 150

```
print('x / y =', x/y)
```

Output: x / y = 1.5

```
print('x % y =', x%y)
```

Output: x % y = 5

```
print('x // y =', x//y)
```

Output: x // y = 1

```
print('x ** y =', x**y)
```

Output: x ** y = 576650390625

RA20 APPT

Assignment

ADDITION, SUBTRACTION, MULTIPLICATION AND DIVISION

1. Write a program for addition, subtraction, multiplication and division of two numbers

ALGORITHM:

STEP 1: Display choice of operation in screen, "Choice 1: Add, 2: Sub, 3: Div, 4: Mul, 5: Quit ".

STEP 2: Receive input choice from the user.

STEP 3: Convert the input to integer (n = int (n1)).

STEP 4: If choice of option is 1 then receive input for A and B, Initiate operation c = A + B.

STEP 5: If choice of option is 2 then receive input for A and B, Initiate operation c = A - B.

STEP 6: If choice of option is 3 then receive input for A and B, Initiate operation c = A / B.

STEP 7: If choice of option is 4 then receive input for A and B, Initiate operation c = A * B.

RA20 APTT

Program:

```
print ("Choice 1: Add, 2: Sub, 3: Div, 4: Mul, 5: Quit")

n1 = input("Enter the choice of operation  : ")
n = int(n1)

if (n == 1):
    print (" You have chosen addition Option :")
    a1 = input(" Enter the value for A :")
    b1 = input(" Enter the value for B :")
    a = int(a1)
    b = int(b1)
    c = a + b
    print ("The Result",a,"+",b,"=",c)
elif (n == 2):
    print (" You have chosen subtraction Option :")
    a1 = input(" Enter the value for A :")
    b1 = input(" Enter the value for B :")
    a = int(a1)
    b = int(b1)
    c = a - b
    print ("The Result",a,"-",b,"=",c)
```

```
elif (n == 3):
    print (" You have chosen division Option :")
    a1 = input(" Enter the value for A :")
    b1 = input(" Enter the value for B :")
    a = int(a1)
    b = int(b1)
    c = a / b
    print ("The Result",a,"/",b,"=",c)

elif (n == 4):
    print (" You have chosen multiplication Option
:")
    a1 = input (" Enter the value for A :")
    b1 = input (" Enter the value for B :")
    a = int (a1)
    b = int (b1)
    c = a * b
    print ("The Result", a,"*", b,"=",c)
elif (n==5):
    print ("You have given a wrong option", n)
    exit ()
```

RA20 APTT

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

RA20 APTT

Comparison Operators

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

RA20 APTT

Comparison Operations

```
x = 8
y = 15
('x>y is',x>y)
```

Output: x > y is False

```
print('x< y is',  x<y)
```

Output: x < y is True

```
print('x == y is', x==y)
```

Output: x == y is False

```
print('x != y is', x!=y)
```

Output: x != y is True

```
print('x >= y is', x>=y)
```

Output: x >= y is False

```
print('x<= y is', x<=y)
```

Output: x <= y is True

Logical Operators

❖Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Logical Operations

```
a = True
b = False
print('a and b is', a and b)
```

Output: a and b is False

```
print('a or b is', a or b)
```

Output: a or b is True

```
print('not a is', not a)
```

Output: not a is False

Identity and Membership Operators

❖Identity : To compare if objects are actually the same object, with the same location

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

```
a1 = 3
b1 = 3
a2 = "Python"
b2 = "Python"

a3 = [4,5,6]
b3 = [4,5,6]

print(a1 is not b1)
```

Output: False

```
print(a2 is b2)
```

Output: True

```
print(a3 is b3)
```

Output: False

Identity and Membership Operators

❖Membership: To test if a sequence is presented memory in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
a = "Python operators"
b = {1:'x',2:'y'}

print("P" in a)
```

Output: True

```
print("python" not in a)
```

Output: False

```
print(1 in b)
```

Output: True

```
print('y' in b)
```

Output: False

Bitwise Operators

❖Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Operator Precedence and Associatively

•**Operator Precedence** determines which operator is performed first in an expression with more than one operators with different precedence.

•**Operators Associatively** is used when two operators of same precedence appear in an expression.

•**Associatively** can be either **Left to Right or Right** to Left.

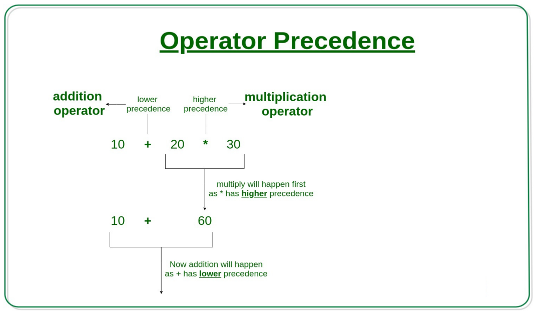
•The operator precedence in Python are listed in the next slide . It is in descending order, upper group has higher precedence than the lower ones.

•To evaluate these type of expressions there is a rule of precedence in Python. It guides the order in which operation are carried out.

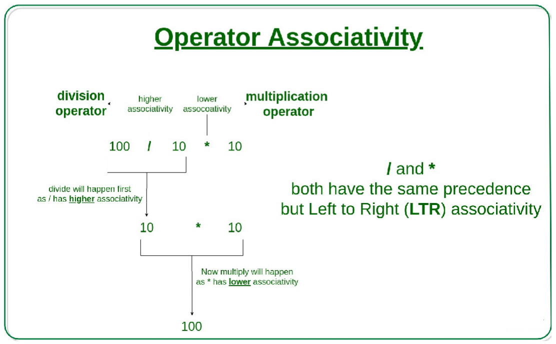
Operator precedence rule in Python

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisions, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

For example: 10 + 20 * 30



10 + 20 * 30 is calculated as **10 + (20 * 30)** and not as **(10 + 20) * 30**


Operator Associativity
division operator
higher associativity
lower assocoativity
multiplication operator
100 / 10 * 10
divide will happen first as / has higher associativity
10 * 10
Now multiply will happen as * has lower associativity
100
/ and *
both have the same precedence
but Left to Right (LTR) associativity

Error Types

The interpreter is designed to execute all valid Python programs.

The interpreter reads the Python source code and translates it into executable machine code(This is the translation phase)

If the interpreter detects an invalid program during the translation phase, it will terminate the program's execution and report an error.

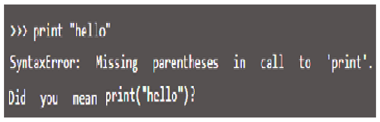
Such errors result from the programmer's misuse of the language.

Syntax Types

❖The most common reason of an error in a Python program is when a certain statement is not in accordance with the prescribed usage.

❖The Python interpreter immediately reports it, usually along with the reason.

❖In Python Print is a built-in function and requires parentheses. The statement above violates this usage and hence syntax error is displayed.



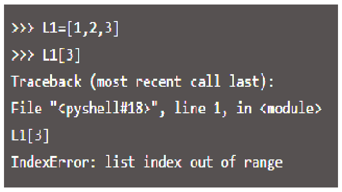
```
>>> print "hello"
SyntaxError: Missing parentheses in call to 'print'.
Did you mean print("hello")?
```

Contd..

❖Many times though, a program results in an error after it is run even if it doesn't have any syntax error.

❖Such an error is a runtime error, called an exception.

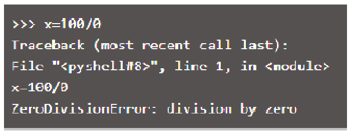
Index Error: is thrown when trying to access an item at an invalid index.



```
>>> L1=[1,2,3]
>>> L1[3]
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
L1[3]
IndexError: list index out of range
```

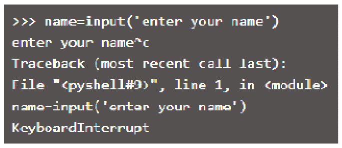
Runtime Error..

Zero Division Error: is thrown when the second operator in the division is zero..



```
>>> x=100/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
x=100/0
ZeroDivisionError: division by zero
```

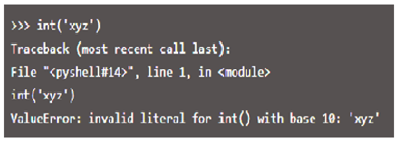
Keyboard Interrupt: when user hits interrupt key during the execution of the program..



```
>>> name=input('enter your name')
enter your name^c
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
name=input('enter your name')
KeyboardInterrupt
```

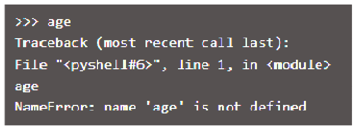
Contd..

Value Error: is thrown when a function's argument is of an inappropriate type.



```
>>> int('xyz')
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
int('xyz')
ValueError: invalid literal for int() with base 10: 'xyz'
```

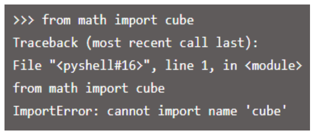
Name Error: is thrown when an object could not be found.



```
>>> age
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
age
NameError: name 'age' is not defined
```

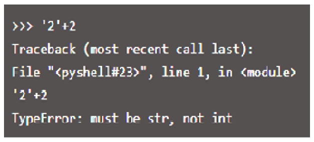
Contd..

Import Error: is thrown when a specified function can not be found..



```
>>> from math import cube
Traceback (most recent call last):
File "<pyshell#16>", line 1, in <module>
from math import cube
ImportError: cannot import name 'cube'
```

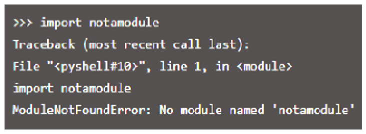
Type Error: is thrown when an operation is applied to an inappropriate type of object



```
>>> '2'+2
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

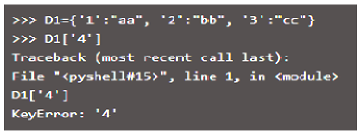
Contd..

Module Not Found Error: is thrown when a module could not be found.



```
>>> import notamodule
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
import notamodule
ModuleNotFoundError: No module named 'notamodule'
```

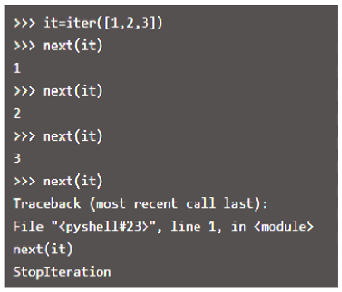
Key Error is thrown when a key is not found..



```
>>> D1={'1':"aa", '2':"bb", '3':"cc"}
>>> D1['4']
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
D1['4']
KeyError: '4'
```

Contd..

Stop Iteration: is thrown when the next() function goes beyond the iterator items.



```
>>> it=iter([1,2,3])
>>> next(it)
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
next(it)
StopIteration
```

Assignment

FIZZ BUZZ PROGRAM

3. Write a program to incorporate FIZZ for any number divisible by 3 and Buzz for any number divisible for 5 and FIZZBUZZ for any number divisible by 3 and 5 as well.

ALGORITHM:

```
STEP 1:  Print "FizzBuzz Program"
STEP 2:  prompt for user input for value N1
STEP 3:  Convert the received value N1 to integer value N
STEP 4:  Create a loop which will execute from 0 to N+1 times.
         If I modules 3 and I modules 5 is zero then
                 Print I value + "= FIZZBUZZ"
         If I modules 3 is zero then
                 Print I value + "= FIZZ"
         If I modules 5 is zero then
                 Print I value + "= Buzz"
         else
                 Print I Value
```

PROGRAM:

```
print ("Fizz Buzz Program :")
n1 = input("Enter the number : ")
n = int(n1)
i = 0
for i in range (n+1):
    if (i % 3 == 0 and i % 5 == 0):
        print (str(i) + "= Fizz Buzz")
    elif (i % 3 == 0):
        print (str(i) + "= Fizz ")
    elif (i % 5 == 0):
        print (str(i) + "= Buzz ")
    else:
        print(i)
```

OUTPUT:

```
FizzBuzz Program
Enter the number: 15
0
1
2
3 = FIZZ
4
5 = BUZZ
6
7
8
9 = FIZZ
10 = BUZZ
11
12 = FIZZ
13
14
15 = FIZZBUZZ
```

Type Conversions

Data Type conversion examples

```
int(10)       => 10        float(10)        => 10.0
int(10.1)     => 10        float(10.1)      => 10.1
int('10')     => 10        float('10')      => 10.0
int('10.1')   => Error     float('10.1')    => 10.1
int('kitten') => Error     float('kitten')  => Error

str(10)       => '10'
str(10.1)     => '10.1'
str('kitten') => 'kitten'
```

Eval Function in Python

- Eval is an interesting hack / utility in Python which lets a Python program run Python code within itself.
- The **eval()** method parses the expression passed to it and runs python expression(code) within the program.
- **The syntax of eval is: eval(expression[, globals[, locals]])**
- The eval() function takes three parameters:
- expression - the string parsed and evaluated as a Python expression
- globals (optional) -a dictionary to specify the available global methods and variables.
- locals (optional)- locals (another dictionary to specify the available local methods and variables.

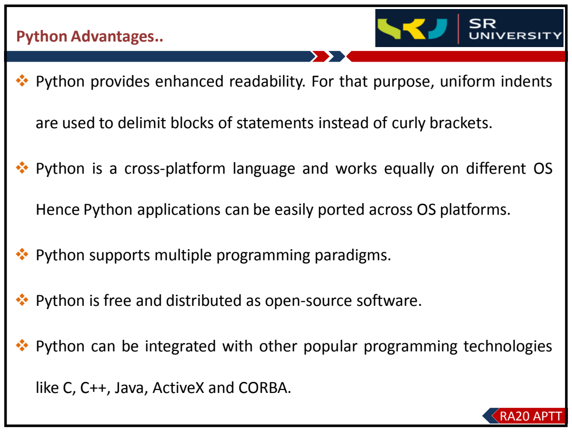
Contd..

```
>>> Eval (" 5 * 4")
>>20
num = 8
square_num =eval ('num * num')
     print(square_num)
>>64
x = 1
 print ( eval('x + 1') )
  2
```

```
>>> eval("2 ** 8")
256
>>> eval("1024 + 1024")
2048
>>> eval("sum([8, 16, 32])")
56
>>> x = 100
>>> eval("x * 2")
200
```

Contd..

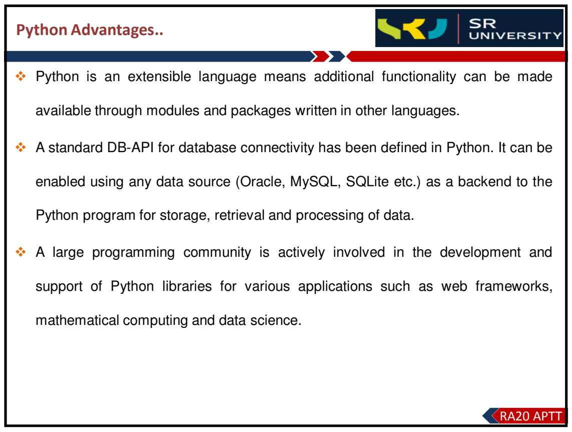
- The name expression for the first argument to eval() highlights that the function works only with expressions and not with **compound statement**
- To evaluate a string-based expression, Python's eval() runs the following steps:
  - Parse expression
  - Compile it to byte code
  - Evaluate it as a Python expression
  - Return the result of the evaluation



Python Advantages..

- Python provides enhanced readability. For that purpose, uniform indents are used to delimit blocks of statements instead of curly brackets.
- Python is a cross-platform language and works equally on different OS Hence Python applications can be easily ported across OS platforms.
- Python supports multiple programming paradigms.
- Python is free and distributed as open-source software.
- Python can be integrated with other popular programming technologies like C, C++, Java, ActiveX and CORBA.

RA20 APTT



Python Advantages..

- Python is an extensible language means additional functionality can be made available through modules and packages written in other languages.
- A standard DB-API for database connectivity has been defined in Python. It can be enabled using any data source (Oracle, MySQL, SQLite etc.) as a backend to the Python program for storage, retrieval and processing of data.
- A large programming community is actively involved in the development and support of Python libraries for various applications such as web frameworks, mathematical computing and data science.

RA20 APTT