OBJECT ORIENTED PROGRAMMING

Strings and files Python

Zuhaib A. Shaikh,
Asst. Prof., CSE Deptt.,QUEST
Web: zuhaib-shaikh.neocities.org

String

- Set of characters, similar as C++ String
- Index starts from zero while, null character isn't observable
- Strings can be assigned and compared using simple operators
- Basic functions:
  - **len**(*string*) – Returns length of string (no. of characters)
  - **.lower**() – Change string's characters into lowercase
  - **.upper**() – Change string's characters into uppercase
  - **.split**(*character_optional*) – Split main string into sub-strings on basis of *character*
  - *Sub-string(optional)***.join**(*string1,string2*) – Combine strings by inserting sub-string in b/w
  - **.replace**(*character1,character2,count_optional*) – Finds and replaces *character1* with *character2* with *count* number of times
  - **.startswith**(*sub-string*) – Returns **True** if strings starts with sub-string
  - **.endswith**(*sub-string*) – Returns **True** if strings ends with sub-string
  - **.find**(*sub-string*) – Returns index of *sub-string* (if available)
  - **.isalpha**() – Returns **True** if all characters are alphabet in string
  - **.isdigit**() – Returns **True** if all characters are digits in string

String

- **.strip**(*String_optional*) – Returns removed characters of *string_optional*, else removed spaces
- **.center**(*width, character_optional*) – Returns string placed in center w.r.t given width and pads *character_optional*, else spaces
- **.rjust**(*width, character_optional*) – Returns string placed in right w.r.t given width and pads *character_optional*, else spaces
- **.ljust**(*width, character_optional*) – Returns string placed in left w.r.t given width and pads *character_optional*, else spaces
- **format**(*String,'width_val'*) – Returns string with width along with the value padded
  - *Charcter_optional*^width – places string at center and pads *character_optional* else spaces
  - *Charcter_optional*>width – places string at right and pads *character_optional* else spaces
  - *Charcter_optional*<width – places string at left and pads *character_optional* else spaces
- **.format**(*variable_name='string_value', …)* – Replaces variable names with their values in the string and return result
- **.count**(*String/character*) – Return the count of *string/character*

```
>>> s1="Python Language"
>>> s1.split()
['Python', 'Language']
>>> ' '.join(['Python', 'Prog'])
'Python Prog'
>>> s1.center(19)
'  Python Language  '
>>> s1.strip()
'Python Language'
>>> format(s1, '*^19')
'**Python Language**'
>>> s="{name} Prog"
>>>s.format(name="Python")
'Python Prog'
>>> s1.lower()
'PYTHON LANGUAGE'
>>> s1.replace('L', 'l')
'Python language'
>>> s1.count('P')
1
```

Filing

- Text or string data stored on permanent storage device
- File is opened in proper mode before applying any operation
  - ***File_handler*** = **open**(*'file_name.extension'*, *mode*)
  - Or using with statement i.e. with **open**(*'file_name.extension'*, *mode*) as ***File_handler***:
  - Modes
    - w – Write
    - r – Read
    - r+ – Read/write
    - a – append
- It is also necessary to close file after performing operators
  - ***File_handler***.**close**()
- Basic functions:
  - .**write**(*String*) – Writes string in the file, escape sequence can be used with it
  - .**writelines**(*Sequence*) – Writes complete sequence in the file
  - .**read**(*Size_optional*) – Returns file contents as string with size of bytes or until EOF
  - .**readline**(*Size_optional*) – Returns string with a line of file contents of specific size or complete line, for loop can also be used to do the same
  - .**next**() – Returns next line from file
  - .**seek**(*handler_location*) – Locates the handler to defined location within file

Directory & Files manipulation

- Files and directories can be manipulated with python **os** library
- Python library or module can be included by writing **import** statement
- Basic functions of **os** library:
  - **.chdir**(*Path*) – Changes working directory to *Path*
  - **.getcwd**() – Returns path of current working directory
  - **.mkdir**(*Path*) – Creates a new directory on *Path*
  - **.rmdir**(*Path*) – Removes directory from given *Path*
  - **.remove**(*Path*) – Remove file from *Path*
- Basic functions of **os.path** sub-library:
  - **.exists**(*Path*) – Returns **True** if *Path* exists
  - **.isdir**(*Path*) – Returns **True** if directory on *Path* exists
  - **.isfile**(*Path*) – Returns **True** if file on *Path* exists
  - **.getsize**(*Path*) – Returns size of the object on *Path*

Exception Handling

- Like C/C++, Python provides exception handling using try-except statements
- Useful when programmer wants to handle run-time errors in their own style
  - E.g.

```
try:
    #do something
Except Error_Identifier_optional as error_msg_handler:
    #do something
```

    - Reacts when specific or any error occurs and provides error message into its handler
- Few error identifiers:
  - **FileNotFoundError** – Error in file opening i.e. file not found
  - **ImportError** – Error in importing library or module
  - **EOFError** – If file hander is at EOF and read operation is performed
  - And many many more
- The errors can also be invoked in the program using **raise** statement

Questions