How to Run Programs

There are multiple ways to tell Python to execute the code we type like...

- system command lines(Intractive prompt)
- icon clicks
- module imports
- exec calls
- menu options in the IDLE GUI etc...

The Interactive Prompt

> simplest way to run Python programs is to type them at Python's interactive command line, sometimes called the interactive prompt.
>
> > most platform neutral way to executing python code.
> >
> > > for running our code on command prompt just wer need to type python at our operating system's prompt.

- Typing the word `python` at our prompt begins an interactive Python session.
- for terminating this session we use...
  - Ctrl-Z for Windows
  - Ctrl-D for Unix
- Anytime we see the >>> prompt, we're in an interactive Python interpreter session.
- here we can type any Python statement or expression and run it immediately.
- if we have not seted our system's PATH environment variable to include Python's install directory, we may need to replace the word `python` with the full path to the Python executable on your machine.
  - for Ex :- C:\Python33\python (for version 3.3)
- Python code which we are going to run in prompt are not generally to be saved in the source files we will be creating.
- because of the interactive session automatically prints the results of expressions we type... we don't usually need to say `print` explicitly at this prompt.
- for printing two or more different statement saperately we need to use print statement in commant prompt as well.
- the `#` part is taken as a comment by Python but may be an error at a system prompt.
- we can also write and run multiline statements at the interactive prompt.
- for writing multiline statements in cmd we need to use \ and ; in our codes.

Why to use an Interactive Prompt?

- The interactive prompt runs code and provides an immdiate results as you go so it is a great place to...
  - experiment with the language
  - test program files.
- The immediate feedback of prompt is often the quickest way to deduce what a piece of code does.

Disadvantage of Intractive prompt

- it doesn't save our code in a file. this means we won't do the bulk of your coding in interactive sessions.
- we can only type Python code at Python's >>> prompt not system commands.for system commands we need to use a os module.
- Don't indent unnested statements at the interactive prompt.
- In the simple shell window interface, the interactive prompt changes to ... instead of >>> for multiline statemens.
- Terminate compound statements at the interactive prompt with a blank line.
- At the interactive prompt, inserting a blank line is necessary to tell interactive Python that you're done typing the multiline statement.
- so we must need to press Enter twice to make a compound statement run.
- blank lines are not required in files and are simply ignored if present.
- If we don't press Enter twice at the end of a compound statement when working interactively, we'll appear to be stuck in a limbo state, because the interactive interpreter will do nothing at all—it's waiting for you to press Enter again!
- also we need to includes blank lines after each compound statement.

System Command Lines and Files

To save programs permanently we need to write our code in files, which are usually known as modules. Modules are simply text files containing Python statements.

Running Files with Command Lines

- we need to type a `python Filename.py` in CMD(or any prompt)
- in order to save the output of a Python script to a file to save it for later use or inspection by while running code in prompt...
  - `python Filename.py > saveit.txt` command is used.

Usage Notes: Command Lines and Files

1. Beware of automatic extensions on Windows and IDLE.

   - If we use the Notepad program to code program files on Windows we need to give the file a .py suffix for save your file explicitly.because Notepad will save your file with a .txt extension.

- Microsoft Word similarly adds a .doc extension by default

2. Use file extensions and directory paths at system prompts but not for imports.
    - Don't forget to type the full name of your file in system command lines. use `python script1.py`
    - Python's import statements omit both the .py file suffix and the directory path (e.g. `import script1`).
3. Use print statements in files.

Clicking File Icons

- we can also able to run Python scripts with file icon clicks(we need to `double click` on mouse).
- a clicked file will be run by one of two Python programs, depending on its extension and the Python you're running.
    - In Pythons 3.2 and earlier...
        - .py files are run by python.exe with a console (CMD) window.
        - .pyw files are run by pythonw.exe files without a console.
    - Byte code files are also run by these programs if clicked.
- By default, Python generates a pop-up console window to serve as a clicked file's input and output.
- If a script just prints and exits console window appears and text is printed there, but the console window closes and disappears on program exit.
- so...Unless we are very fast, or our machine is very slow, we won't get to see your output at all.
- If you need your script's output to stick around when we launch it with an icon click, simply put a call to the built-in input function at the very bottom of the script in 3.X.

Icon-Click Limitations

- when we run a python file by clicking file icons we also may not get to see Python error messages.
- If our script generates an error, the error message text is written to the pop-up console window which then immediately disappears.
- Because of these limitations, it is best to run codes by icon clicks after they have been debugged, or have been instrumented to write their output to a file and catch and process any important errors.
- we can also able to get error message using exception Handling.

Module Imports and Reloads

- import is also a way to launch python programs
- every file of Python source code whose name ends in a .py extension is a module.
- No special code or syntax is required to make a file a module.
- Other files can access the items a module defines by importing that module.
- import operations essentially load another file and grant access to that file's contents.
- we can run the `Filename.py` file we created earlier with a simple import...
    - python import Filename
    - here... we don't need to put extension .py
- This works only once per session by default. After the first import later imports do nothing, even if you change and save the module's source file again in another window.
- If we really want to force Python to run the file again in the same session without stopping and restarting the session, we need to instead call the reload function available in the imp standard library

module.
    - from imp import reload # Must load from module in 3.X
    - reload(script1)

- The reload function expects the name of an already loaded module object so we has to have successfully imported a module once before we reload it.
- if the import reported an error, you can't yet reload and must import again.
- Notice that reload also expects parentheses around the module object name, whereas import does not.
- reload is a function that is called, and import is a statement.
- we must pass the module name without extension to reload as an argument in parentheses

Difference Between import and from-import

module files usually define more than one name to be used in and outside the files. for Ex. here is a file named `variables.py`

In [1]:

```
a = 'dead' # Define three attributes
b = 'parrot' # Exported to other files
c = 'sketch'
print(a, b, c)
```

```
dead parrot sketch
```

if we use only `import` then we get a module with attributes but if we use `from` we get copies of the file's names.

- python import variables # loads the whole module and access its all attributes.
- python import variables.b # loads the whole module and access its attribute b.
- python from variables import c # not load whole module instead it Copies only c name out and don't load b and a.

Namespaces

- Python programs are composed of multiple module files linked together by import statements and each module file is a package of variables that is called as a namespace.
- each module is a self-contained namespace.
- one module file cannot see the names defined in another file unless it explicitly imports that other file. Because of this, modules serve to minimize name collisions in our code because each file is a self-contained namespace the names in one file cannot clash with those in another even if they are spelled the same way.

Limitations of import and reload

- we only able to run same file after applying changes using import method by reload function but reloads aren't transitive.

- reloading a module reloads that module only, not any modules it may import so you sometimes have to reload multiple files.
- if we imported more than one module than we need to reload each of them seperately.

Using exec to Run Module Files

- The exec(open('module.py').read()) built-in function call is another way to launch files from the interactive prompt without having to import and later reload.
- Each such exec runs the current version of the code read from a file, without requiring later reloads python exec(open('script1.py').read())

Limitations of exec

- exec overwrites variables you may currently be using. For example, our script1.py assigns to a variable named x = 'Spam!'. If that name is also being used in the place where exec is called, the name's value is replaced.

In [2]:

```
x = 999
print('value of x before exec :- ', x)
exec(open('file.py').read()) # Code run in this namespace by default
print('final value of x is :- ', x)  # Its assignments can overwrite names here
```

```
value of x before exec :-  999
x in file :-  76
y in file :-  43
final value of x is :-  76
```

- but basic import statement runs the file only once per process and it makes the file a separate module namespace so that its assignments will not change variables in your scope.

IDLE User Interface

- IDLE provides a graphical user interface for doing Python development and it's a standard and free part of the Python system.
- IDLE is usually referred to as an integrated development environment (IDE) because it binds together various development tasks into a single view.
- IDLE is a desktop GUI that lets you edit, run, browse, and debug Python programs all from a single interface.
- in Windows IDLE comes installed automatically with standard Python.
- we able to launch IDLe by a search for "idle" by browsing your "All apps" Start screen display or by using File Explorer to find the idle.py.
- we can also able to launch IDLE By this command `python -m idlelib.idle` in cmd. # Run idle.py in a package on module path.
- IDLE works like all interactive sessions code we type. it's run immediately after you type it and serves as a testing and experimenting tool.
- it also displays an errors to help us to spot mistakes.

Features of IDLE

- Auto-indent and unindent for Python code in the editor
- Word auto-completion while typing, invoked by a Tab press
- Balloon help pop ups for a function call when you type its opening “(”
- Pop-up selection lists of object attributes when you type a “.” after an object’s name and either pause or press Tab.
- IDLEprovides more advanced features, including a point-and-click program graphical debugger and an object browser.
- The IDLE debugger is enabled via the Debug menu and the object browser via the File menu.

Limitations of IDLE

1. we must need to add “.py” explicitly when saving our files.
2. Run scripts by selecting Run→Run Module in text edit windows not by interactive imports and reloads.because un→Run Module menu option in IDLE always runs the most current version of our code file.
3. There is currently no clear-screen option in IDLE.so sometimes it gets too much messi code.
4. tkinter GUI and threaded programs may not work well with IDLE.
5. If connection errors arise try starting IDLE in single-process mode.using `python -m idlelib.idle -n` command we able to start IDLE in single-process mode from anywhere.

Other Popular IDLEs

1. Eclipse and PyDev
    - Eclipse is an advanced open source IDE GUI. Originally developed as a Java IDE.Eclipse also supports Python development when you install the PyDev plug-in.
2. Komodo
    - A full-featured development environment GUI for Python.Komodo includes standard syntax coloring, text editing, debugging, and other features
3. NetBeans IDE for Python
    - NetBeans is a powerful open source development environment GUI with support for many advanced features for Python developers. code completion, automatic indentation and code colorization, editor hints, code folding, refactoring, debugging, code coverage and testing, projects, and more. It may be used to develop both CPython and Jython code.
4. Visual Studio
5. PyCharm
6. Spyder
7. Jupyter Notebook

How to Debug a Python code?

1. Do nothing

- when you make a mistake in a Python program you get a very useful and readable error message.
- read the error message and go fix the tagged line and file.
- It may not always be ideal for larger systems you didn't write though.

2. Insert print statements
   - main way that Python programmers debug their code is to insert print statements and run again.
   - Because Python runs immediately after changes this is usually the quickest way to get more information than error messages provide.
   - Just remember to delete or comment out the debugging prints before you ship your code.
3. Use IDE GUI debuggers
   - For larger systems you didn't write and for beginners who want to trace code in more detail most Python development GUIs have some sort of point-and-click debugging support.
4. Use the pdb command-line debugger
   - For ultimate control Python comes with a source code debugger named pdb.which is available as a module in Python's standard library.
   - In pdb you type commands to step line by line, display variables, set and clear breakpoints, continue to a breakpoint or error, and so on.
   - You can launch pdb interactively by importing it or as a top-level script.
   - pdb also includes a postmortem function (pdb.pm()) that you can run after anexception occurs to get information from the time of the error.
5. Use Python's –i command-line argument
   - Short of adding prints or running under pdb you can still see what went wrong on errors.
   - If we run our script from a command line and pass a -i argument between python and the name of our script Python will enter into its interactive interpreter mode For Ex. python –i m.py
   - when our script exits whether it ends successfully or runs into an error. At this point you can print the final values of variables to get more details about what happened in your code because they are in the top-level namespace.
6. Other options
   - The Winpdb system is a standalone debugger with advanced debugging support and cross-platform GUI and console interfaces.