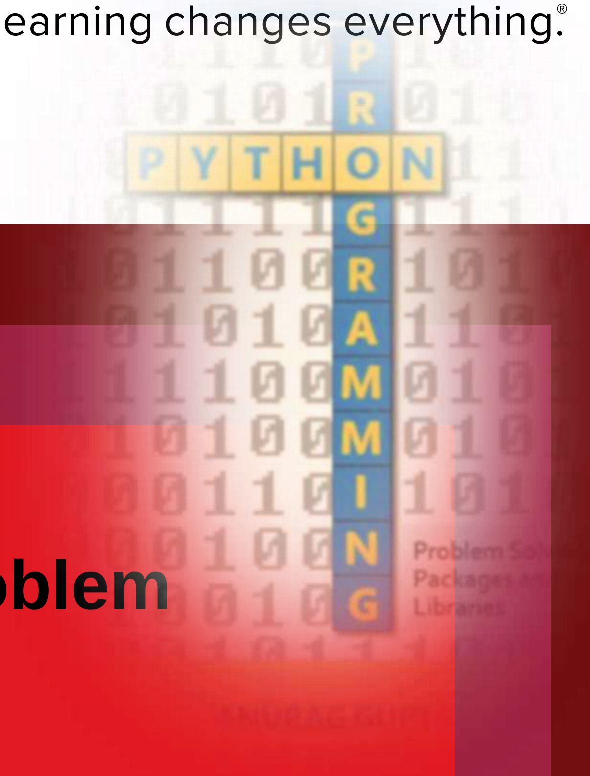
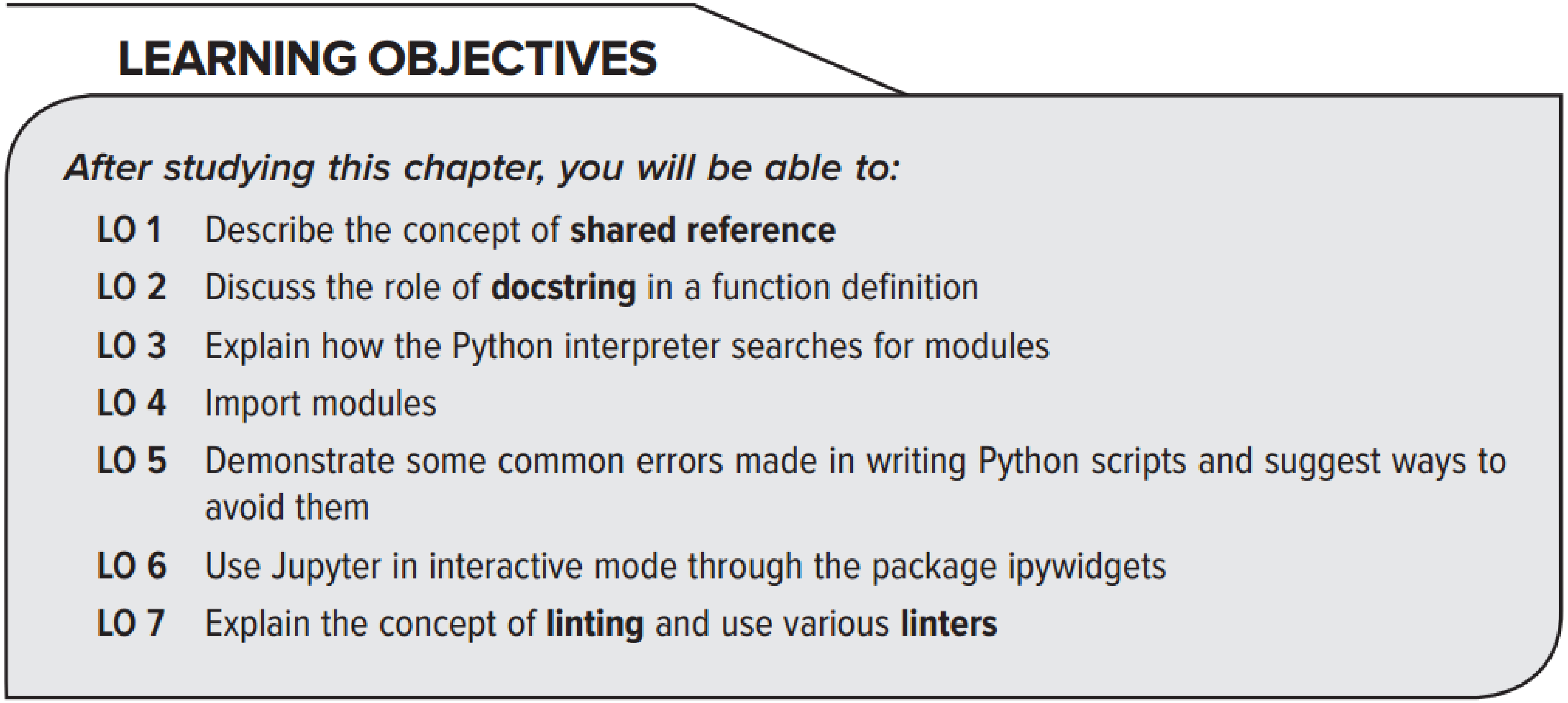
Because learning changes everything.®



TITLE: Python Programming: Problem Solving, Packages and Libraries

Lecture PPT
Edition

Some Additional Advanced Topics: Learning Objectives



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 1 Describe the concept of **shared reference**
- LO 2 Discuss the role of **docstring** in a function definition
- LO 3 Explain how the Python interpreter searches for modules
- LO 4 Import modules
- LO 5 Demonstrate some common errors made in writing Python scripts and suggest ways to avoid them
- LO 6 Use Jupyter in interactive mode through the package ipywidgets
- LO 7 Explain the concept of **linting** and use various **linters**

Shared reference and in-place change. (Relating to function calls)

- In Python, all objects are either mutable or immutable.
- Mutable objects can be changed while the immutable cannot.
- Immutable objects in Python are integers, strings, tuples, and so on. If you try to change an immutable object, it will either create a new object or give an error message.
- Mutable objects in Python are lists, dictionaries and sets. Mutable objects can be changed in-place.
- Therefore, a list (being mutable) can be changed in-place, but strings (being immutable) cannot be changed in-place.

Shared reference, equality and sameness

As pointed out earlier, objects in Python have tags or references through which these objects can be accessed. Objects in Python have four important characteristics, which are (i) identity (ii)type (iii) value and (iv) scope.

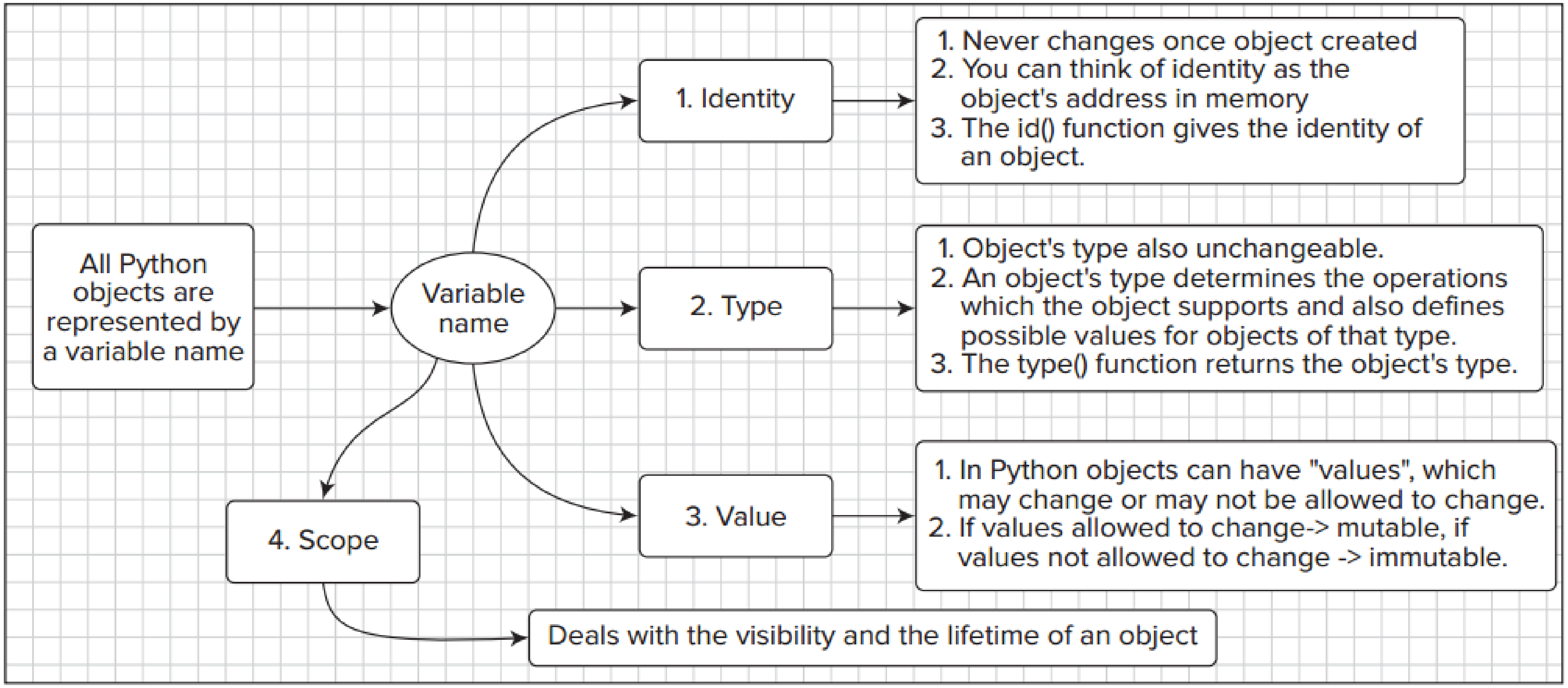

All Python objects are represented by a variable name
Variable name
1. Identity
1. Never changes once object created
2. You can think of identity as the object's address in memory
3. The id() function gives the identity of an object.
2. Type
1. Object's type also unchangeable.
2. An object's type determines the operations which the object supports and also defines possible values for objects of that type.
3. The type() function returns the object's type.
3. Value
1. In Python objects can have "values", which may change or may not be allowed to change.
2. If values allowed to change-> mutable, if values not allowed to change -> immutable.
4. Scope
Deals with the visibility and the lifetime of an object

FIGURE 12.1 Four important characteristics of a variable

Docstring (In function definition) -- 1

1. **Docstring is the first string after the function header in the function definition. Docstring is short for document string.**
2. **Docstrings are used to explain to the end user of the function, as to what the function does, not how. Docstrings are optional, but should be given to help the user of the code.**
3. **In any program, proper documentation is important and conventions are laid down as to how documentation is to be done. There are a number of ways in which documentation of programs is done. Some examples are: readme files, comments and docstrings. Not all programming languages provide docstring but Python does. In Python, docstrings are just like comments but the purpose of writing docstrings is different from the purpose for comments.**
4. **Docstrings are for those who want to USE the code written by a particular person.**
5. **However, comments are for those who want to UNDERSTAND or EXTEND the code written.**

Docstring (In function definition) -- 2

6. Python provides an in-built attribute called __doc__ to every function class and method. When you say automatically provides, it means that the moment you create a function, method or a class, the Python interpreter on its own creates an attribute named __doc__ for that function, method or class.

7. Further this __doc__ attribute points to the docstring of that function, method or class. So, if you write a function say, my_func and a user wants to know what this function does, then all he has to do is see print(my_func.__doc__), and he will get the docstring of that function on his screen.

8. Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods.

9. Note that the comments using has, that is, (#) cannot be accessed using the __doc__ attribute. The reason that comments, that is, those beginning with hash are not included in attribute __doc__ is that comments are not meant to explain what a function does, rather, comments are meant to explain how a function works. So docstrings are for users and comments are for programmers.

How to access the docstring of a function/ method/ class etc

The docstring of an inbuilt function can be accessed in two ways:

1. **By using the __doc__ attribute of the function and**
2. **By using the help(functionName) function in Python**

Section 12.3 of book -- 1

(Concepts related to python module, __name__ attribute and virtual environment.)

This section of the book covers topics like:

- **__name__ attribute**
- **How Python interpreter searches for modules. (Relating to import statements)**
- **Problems that may arise in importing modules**
- **Importing only some of the attributes:-**
- **Using import ***
- **Attributes with leading underscore and import**
- **Using __file__ attribute to find location of an imported module:**
- **Using reload()**
- **Understanding the use of the command line**

All these topics are covered very well in the book and are also quite straight forward and hence not discussed in this presentation.

Section 12.3 of book -- 2

(Concepts related to python module, __name__ attribute and virtual environment.)

However the following topics are covered in this presentation

- Running a Python script from the command line
- How optional parameters may be passed to the script, which is being executed
- Getting the 'dependency' tree of a module
- Virtual environment

Running a Python script from the command line -- 1

To run a Python script from the command line, the format is as follows:

```
$ python filename.py arg1 arg2 arg3 … argN
```

In the above script:

1. **$ represents the command prompt. It may vary on different OS. In windows, it is >.**

2. **The word Python indicates that the program to be used is the Python interpreter. Note that the OS must be able to find the location of python.exe file. To do this, the full path of this exe file must be included in the path environmental variable on a Windows machine. The other option is that this command is executed from within the folder containing the python.exe file.**

3. **The parameter filename.py indicates the Python script to be executed. Here you have two options as follows:**

4. **Finally, the other arguments, that is, arg1 arg2 arg3 … argN are some of the other optional arguments that may be given on the command line.**

Running a Python script from the command line -- 2

5. But the Python script (which is being executed) must know how to use these optional parameters.
6. To literally catch these optional parameters, Python has an attribute called argv in the sys module. Therefore, sys.argv gives a list of strings representing the arguments (as separated by spaces) on the command line.
7. Note that sys.argv is a Python list and like all lists begins at index 0. The index 0, that is, sys.argv[0] is reserved for the executable file name. The other arguments beginning with 1 are reserved for the optional command line parameters. So, if no optional parameters are passed, then there will be only sys.argv[0], but if say two command line parameters are passed on the command line, then they will become available to the script as sys.argv[1] and sys.argv[2] respectively. (Note that the strange name argv stands for argument value and comes from the C programming language.)

Many Python packages are dependent on other packages. So, when installing Python packages, you should know its dependencies. One easy way to know dependencies of “installed” packages is to use the pipdeptree utility. The pipdeptree utility works on the command line. It can show the installed Python packages in the form of a dependency tree. You can install pipdeptree with a pip command as shown:

```
pip install pipdeptree
```

Now you can run this command on the command prompt. Once you do so, you will get a dependency tree of all your Python modules. This

```
$pipdeptree
```

Virtual environment

- A virtual environment is a method whereby you can have different versions and different instances of Python running on the same machine.
- It is a way to run different versions of Python for different projects.
- Therefore, by having virtual environments you can have different versions of Python and different versions of packages of python on the same machine.
- Related to the concept of virtual environment , is the concept of dependencies.
- If a Python library has been built-upon other libraries, then those libraries must be installed for this package to work properly.
- So, if you have a virtual environment, then you can download and install the 'dependencies' as per your need, that is, the need of a project.

The topic of virtual environment is discussed in details in the book. Please refer to it.

ipywidgets

The Anaconda package has a pre bundled package called ipywidgets, which enables the Jupyter notebook to be used in the interactive mode.

The text book has a detailed discussion on this module ipywidgets. Hence it is not included in this presentation

Linting in python

A linter is a tool, which analyses the source code and reports on some or all of the following:

- **Variations/ violations of coding style (PEP8 for python)**
- **Bugs or design flaws**
- **Code patterns which may be dangerous or introduce vulnerabilities.**

Another thing to note is that a linter does static analysis of your code. This means that the code is analysed but not executed. So, if there is a program which may have some dangerous consequences, it would be better to first do linting on it rather than run it because linting does not cause the code to be executed.

Linting packages available

- **A large number of linting packages are available.**
- **Here, only two namely pycodestyle and pylint are discussed.**
- **Pycodestyle is for PEP8 confirmation and pylint does many other things.**

pycodestyle

- Pycodestyle (Formerly PEP8) is the linter tool to check the Python code against the style conventions of PEP8
- Pycodestyle generates errors (beginning with E) or warnings (beginning with W).
- Following table shows some common errors and warnings generated by pycodestyle

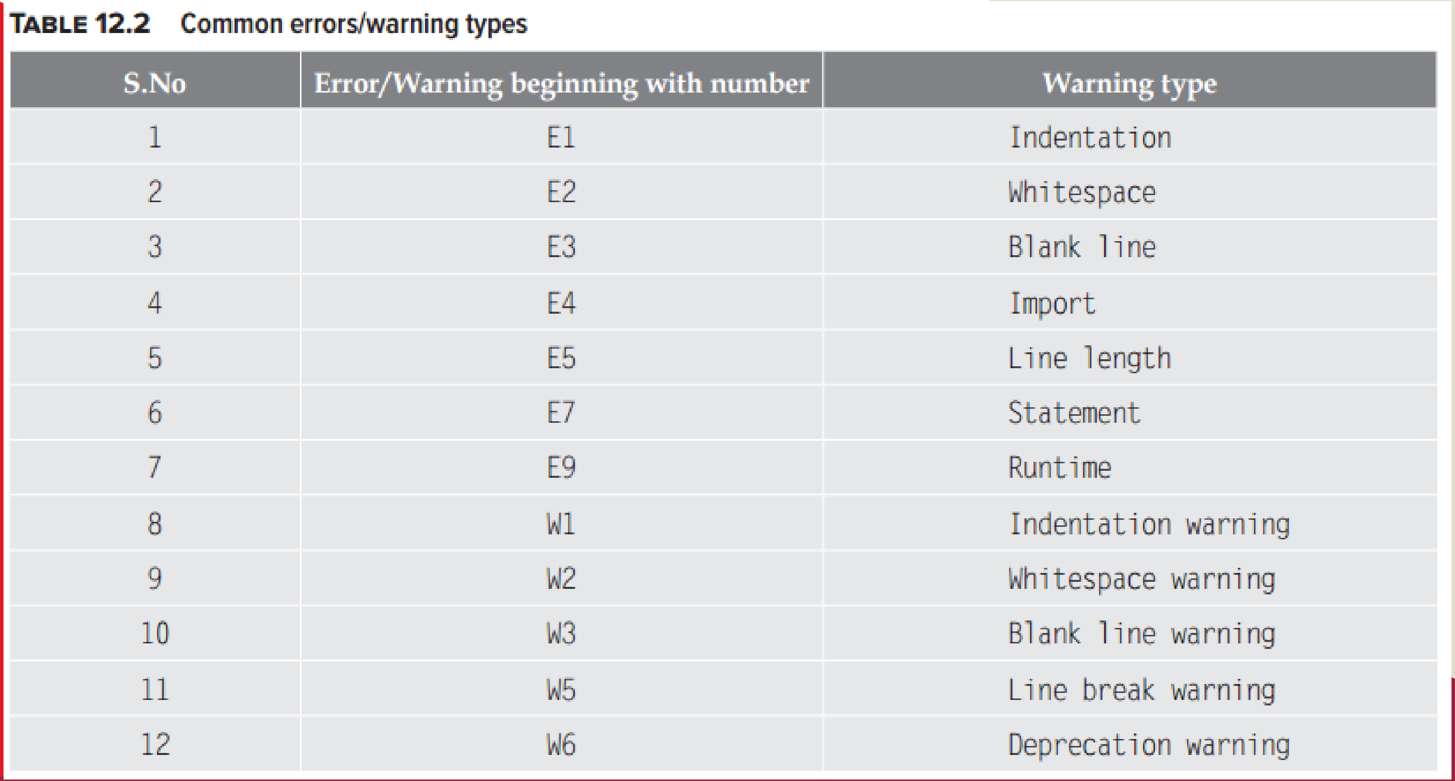


TABLE 12.2 Common errors/warning types

S.No	Error/Warning beginning with number	Warning type
1	E1	Indentation
2	E2	Whitespace
3	E3	Blank line
4	E4	Import
5	E5	Line length
6	E7	Statement
7	E9	Runtime
8	W1	Indentation warning
9	W2	Whitespace warning
10	W3	Blank line warning
11	W5	Line break warning
12	W6	Deprecation warning

pylint

Pylint is used to check for errors in the Python script. While pycodestyle only checks for code writing style, pylint checks not only for style but also errors.

Pylint checks for both bugs as well as quality level. Some things it checks are as follows:

- **It checks for line length. Line length is the number of characters contained in a line of code. Good programming practices recommend to keep it to 80. If it goes beyond 80, pylint will point it out, checking the length of each line.**
- **It checks whether the variable names used by the programmer are in conformity with PEP8 guidelines. For instance, PEP8 recommends that you should name constants with all upper case. So PI = 3.14 is good but pi = 3.14 is not as per PEP8.**
- **Checking whether the declared interfaces are truly implemented. (This is an advanced concept and therefore not discussed here.).**

Messages by pylint

Pylint gives five types of messages as follows:

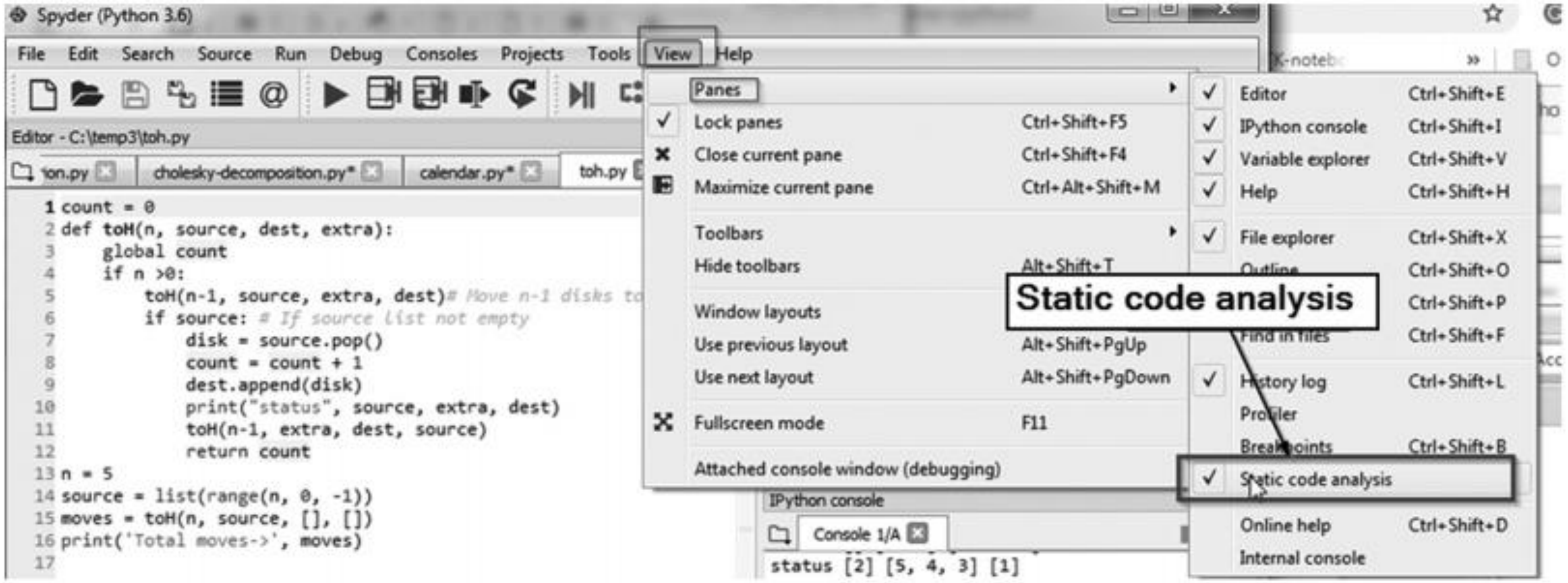
```
There are 5 kind of message types :
    * (C) convention, for programming standard violation
    * (R) refactor, for bad code smell
    * (W) warning, for python specific problems
    * (E) error, for probable bugs in the code
    * (F) fatal, if an error occurred which prevented pylint from
           doing further processing.
```

Pylint and static code analysis in Spyder

Pylint is already integrated in Spyder and also some other IDE (Like Visual Studio and Visual Studio Code).

For other IDE (Like Eclipse and PyDev, you have to integrate it manually).

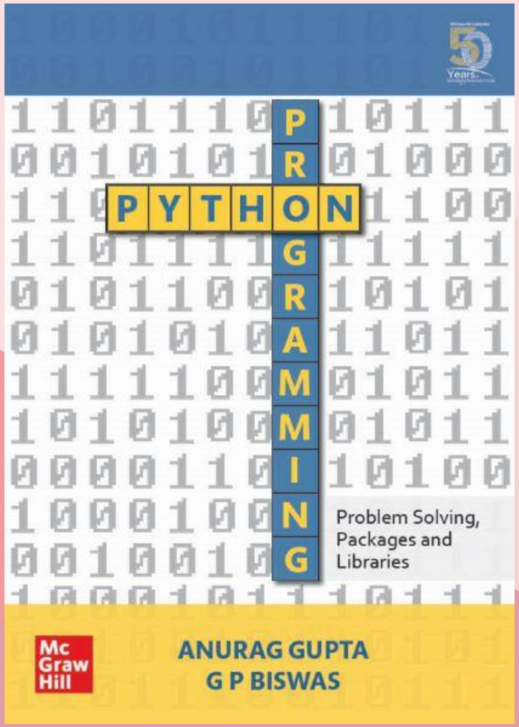
You can see the Static code analysis window on Spyder by doing the following: View > Panes > Static code analysis.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\temp3\toh.py
ton.py
cholesky-decomposition.py*
calendar.py*
toh.py
Panes
Lock panes Ctrl+Shift+F5
Close current pane Ctrl+Shift+F4
Maximize current pane Ctrl+Alt+Shift+M
Toolbars
Hide toolbars Alt+Shift+T
Window layouts
Use previous layout Alt+Shift+PgUp
Use next layout Alt+Shift+PgDown
Fullscreen mode F11
Attached console window (debugging)
Editor Ctrl+Shift+E
IPython console Ctrl+Shift+I
Variable explorer Ctrl+Shift+V
Help Ctrl+Shift+H
File explorer Ctrl+Shift+X
Outline Ctrl+Shift+O
Ctrl+Shift+P
Find in files Ctrl+Shift+F
History log Ctrl+Shift+L
Profiler
Breakpoints Ctrl+Shift+B
Static code analysis
Online help Ctrl+Shift+D
Internal console
Static code analysis
IPython console
Console 1/A

```
count = 0
def toH(n, source, dest, extra):
    global count
    if n >0:
        toH(n-1, source, extra, dest)# Move n-1 disks to
        if source: # If source list not empty
            disk = source.pop()
            count = count + 1
            dest.append(disk)
            print("status", source, extra, dest)
            toH(n-1, extra, dest, source)
            return count
n = 5
source = list(range(n, 0, -1))
moves = toH(n, source, [], [])
print('Total moves->', moves)
```

```
status [2] [5, 4, 3] [1]
```

FIGURE 12.21 Screenshot showing how the "Static code analysis" window can be opened on Spyder IDE


PYTHON
PROGRAMMING
Problem Solving, Packages and Libraries
Mc Graw Hill
ANURAG GUPTA
G P BISWAS

Thank You!

For any queries or feedback contact us at:

support.india@mheducation.com

1800-103-5875

www.mheducation.co.in