PRG1

W
E
E
K

2

NGEE ANN
SCHOOL OF INFOCOMM TECHNOLOGY

String Formatting & Debugging

Programming I (PRG1)

Diploma in Information Technology

Diploma in Financial Informatics

Diploma in Cybersecurity & Digital Forensics

Common ICT Programme

Year 1 (2019/20), Semester 1

Objectives

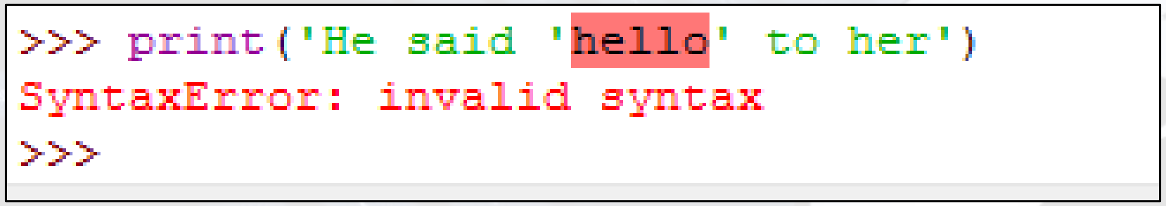
At the end of this lecture, you will learn about….

- ❑ Escape Sequence
- ❑ String Formatting
- ❑ Debugging

Escape Sequences

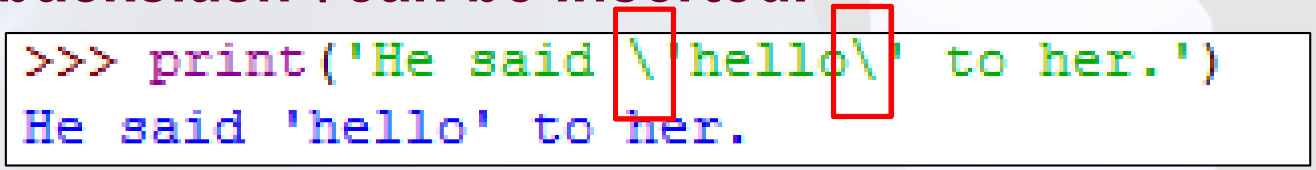
Escape Sequence

- Consider the case we wish to print out the literal quotes ' ' as part of a string. A syntax error will be produced.



```
>>> print('He said 'hello' to her')
SyntaxError: invalid syntax
>>>
```

- To help us 'escape' single quotes or double quotes, a backslash \ can be inserted.



```
>>> print('He said \'hello\' to her.')
He said 'hello' to her.
```

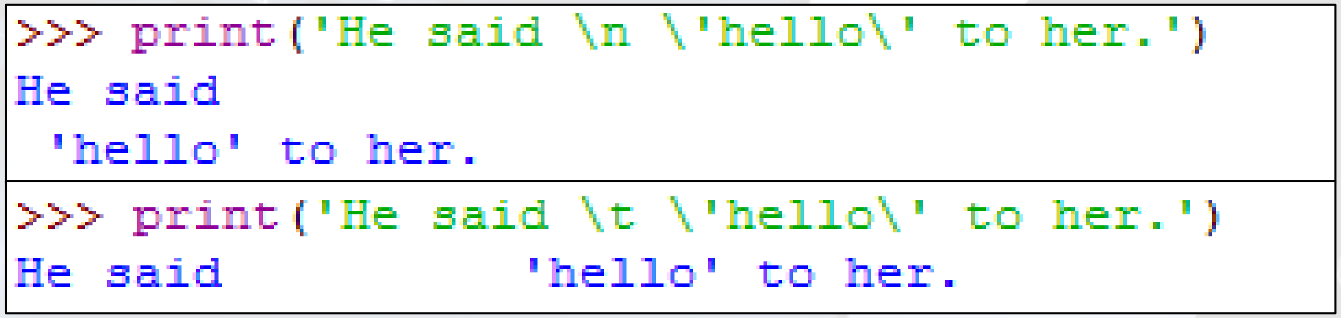
- The backslash character together with an *escape character* form an *escape sequence*.

Escape Sequence

- There are other escape sequences but the two most commonly used are:

  \n ↲ new line

  \t ⇥ tab stop



```
>>> print('He said \n \'hello\' to her.')
He said
 'hello' to her.
```

```
>>> print('He said \t \'hello\' to her.')
He said           'hello' to her.
```

Refer to: https://docs.python.org/3/reference/lexical_analysis.html for full table of escape sequences.

String Formatting

String Formatting

- Python provides 2 advanced ways to do *String Formatting* – allowing multiple substitutions in a string
  - ✓ Using string formatting operator %

    ```
    '...%s...'% (arguments)
    ```

    - %s are placeholders to be replaced by the arguments.
  - ✓ Using string formatting method call format()

    ```
    '...{}...'.format(arguments)
    ```

    - { } in original string are placeholders to be replaced by respective arguments.
- 2nd method is the preferred standard in Python 3 (https://docs.python.org/2/library/stdtypes.html#str.format)

String Formatting

- Basic Formatting

```
>>> '{} {}'.format('one', 'two')
'one two'
```

```
>>> '{} {}'.format(1, 2)
'1 2'
```

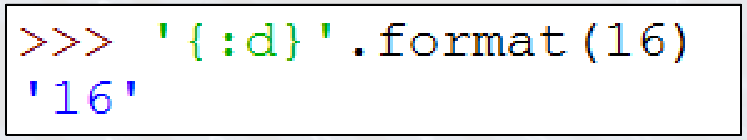
- **format()** method allows rearrangement in output without changing order of arguments:

```
>>> '{1} {0}'.format('one', 'two')
'two one'
```

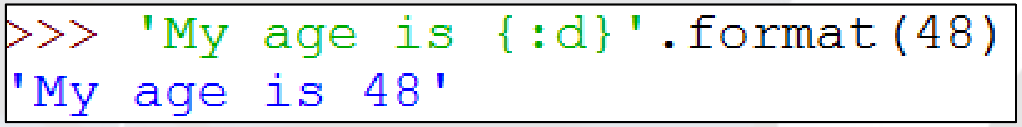
```
>>> '{2} {3} {5} {1} {4} {6}'.format('See', 'how', 'the', '
words', 'are', 'mixed', 'up')
'the words mixed how are up'
```

String Formatting

- Integers

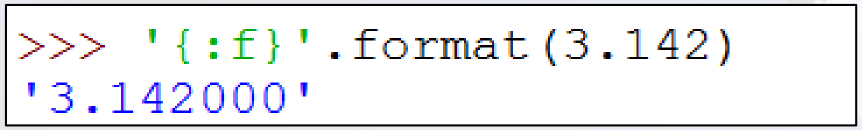


```
>>> '{:d}'.format(16)
'16'
```

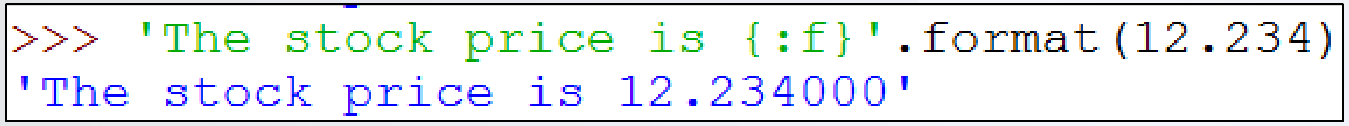


```
>>> 'My age is {:d}'.format(48)
'My age is 48'
```

- Floating Point



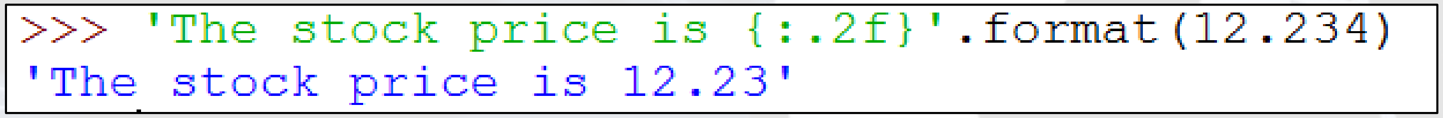
```
>>> '{:f}'.format(3.142)
'3.142000'
```



```
>>> 'The stock price is {:f}'.format(12.234)
'The stock price is 12.234000'
```

String Formatting

- The default is to have six decimal points of *precision* for float.
- The precision can be changed as follows:



```
>>> 'The stock price is {:.2f}'.format(12.234)
'The stock price is 12.23'
```

String Formatting

- Strings

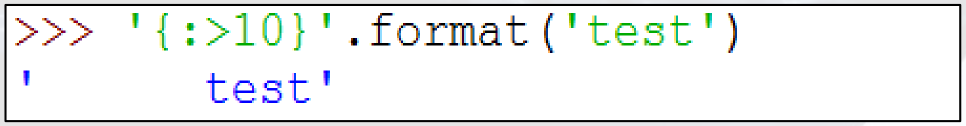
```
>>> name = 'Mandy'
>>> greeting = 'How are you?'
>>> 'Hello {:s}, {:s}'.format(name, greeting)
'Hello Mandy, How are you?'
```

- Padding and Alignment
  - ✓ By default, values will take up as many characters as needed to represent the content.
    - possible to pad a value to a certain length.
- format() defaults alignment to left for strings.
  - ✓ Note: alignment to right for other types

```
>>> '{:10}'.format('test')
'test      '
```

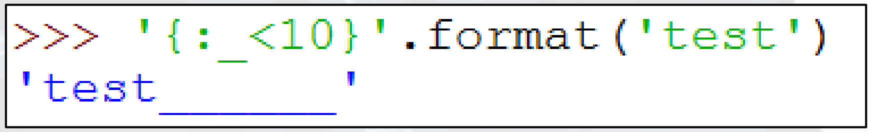
String Formatting

- format() allows using **<** or **>** to denote direction of alignment

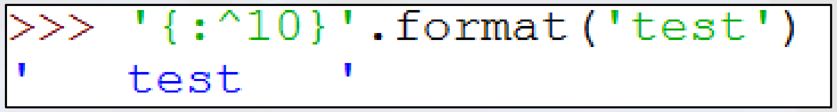


```
>>> '{:>10}'.format('test')
'      test'
```

- format() allows choosing of character to do the padding:

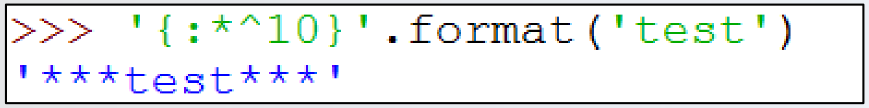


```
>>> '{:_<10}'.format('test')
'test______'
```



```
>>> '{:^10}'.format('test')
'   test   '
```

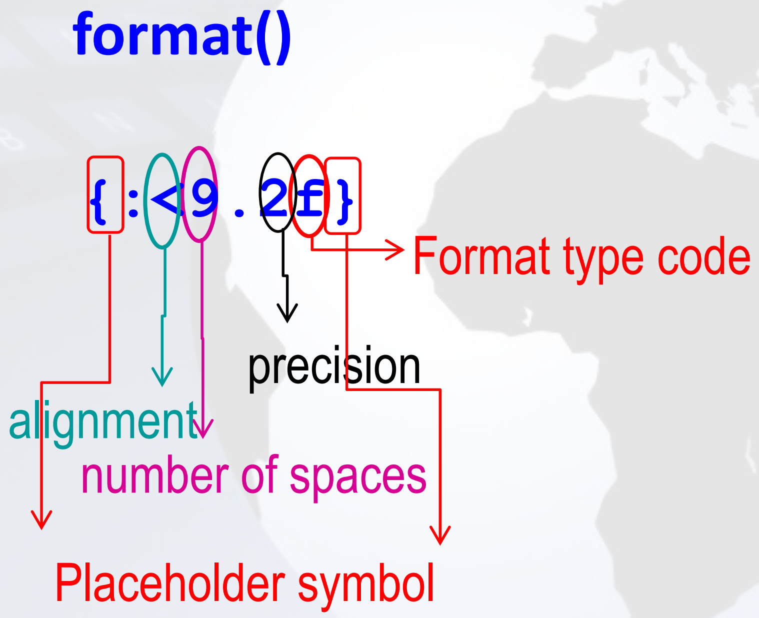
- format() allows specifying of center alignment:



```
>>> '{:*^10}'.format('test')
'***test***'
```

String Formatting

- format() make use of a format instruction called ***Format Specifier***



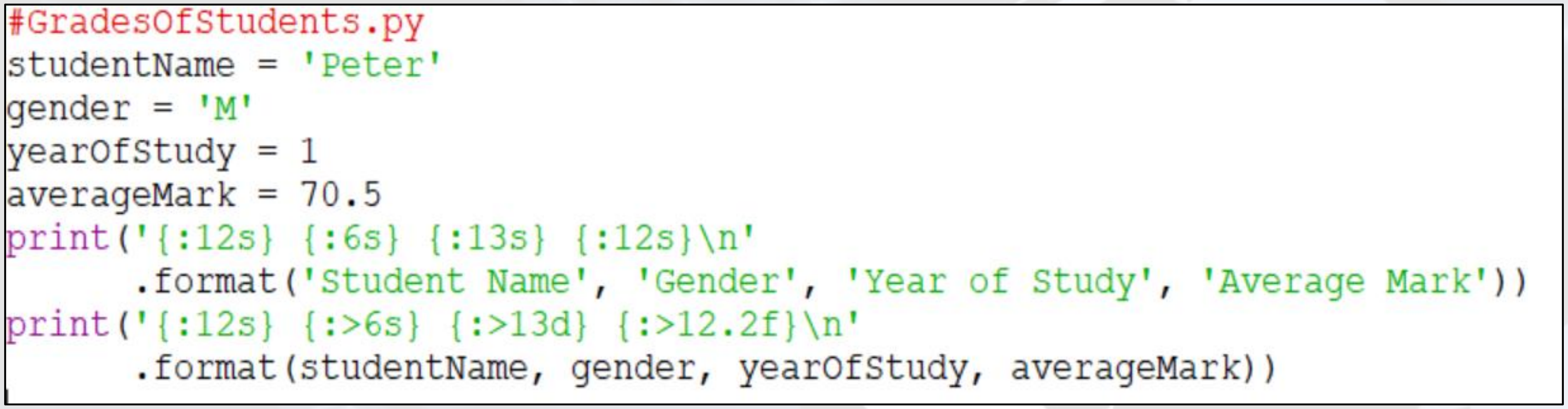
String Formatting

- Commonly used Format Type Codes:

Format type code	Description
`s`	string
`c`	character
`d`	decimal (base 10) integer
`o`	octal integer
`x`	hex integer
`X`	same as x, but uppercase
`f`	floating point real numbers
`e`	exponent notation

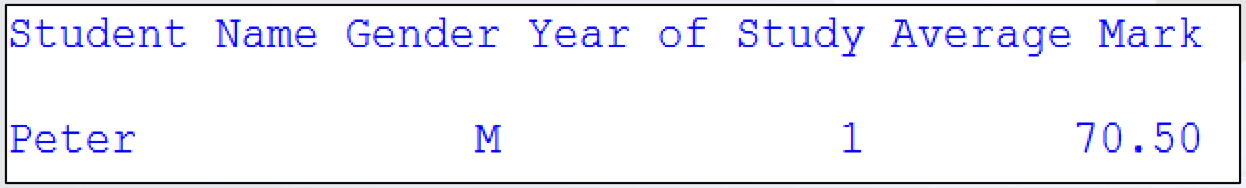
String Formatting

Application:



```python
#GradesOfStudents.py
studentName = 'Peter'
gender = 'M'
yearOfStudy = 1
averageMark = 70.5
print('{:12s} {:6s} {:13s} {:12s}\n'
      .format('Student Name', 'Gender', 'Year of Study', 'Average Mark'))
print('{:12s} {:>6s} {:>13d} {:>12.2f}\n'
      .format(studentName, gender, yearOfStudy, averageMark))
```

Output:

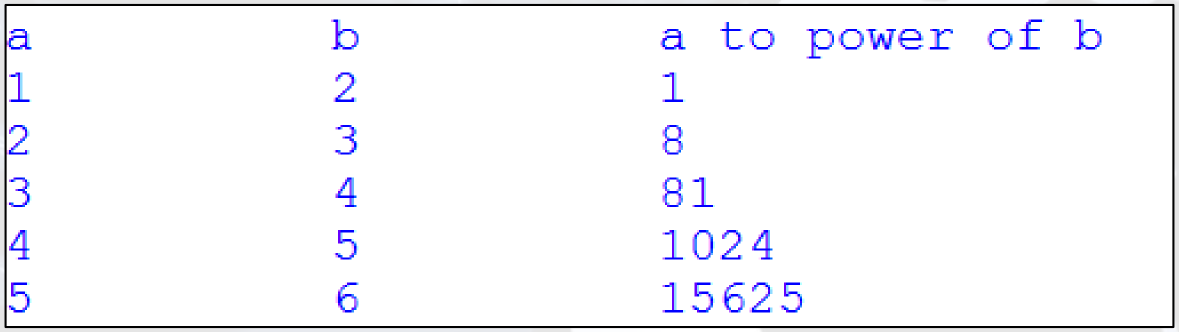


```
Student Name Gender Year of Study Average Mark

Peter             M             1        70.50
```

Activity 1

Write a program that displays the following table:



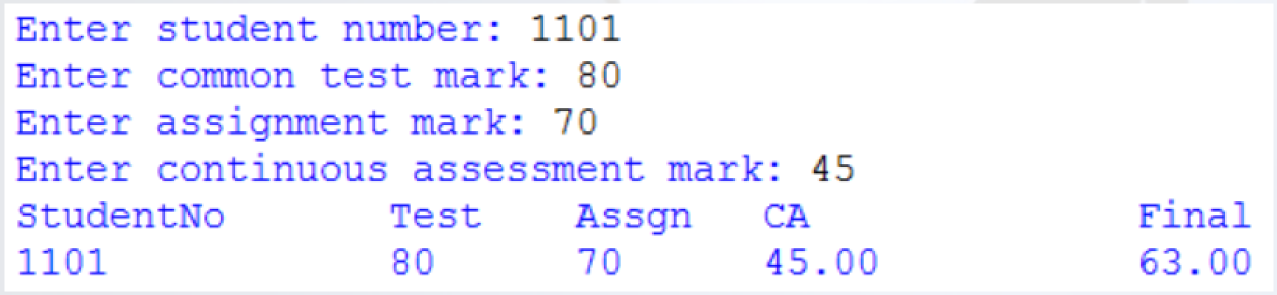
```
a          b          a to power of b
1          2          1
2          3          8
3          4          81
4          5          1024
5          6          15625
```

Activity 2

Recall in Week 1 (Part 2) Activity 3:

The final mark for PRG1 module is calculated based on 30% of common test, 30% of assignment and 40% of continuous assessment.

Now modify the program to input student id, the marks for common test, assignment and continuous assessment and display the results in the following tabular format:



```
Enter student number: 1101
Enter common test mark: 80
Enter assignment mark: 70
Enter continuous assessment mark: 45
StudentNo       Test    Assgn   CA              Final
1101            80      70      45.00           63.00
```

Types of Errors

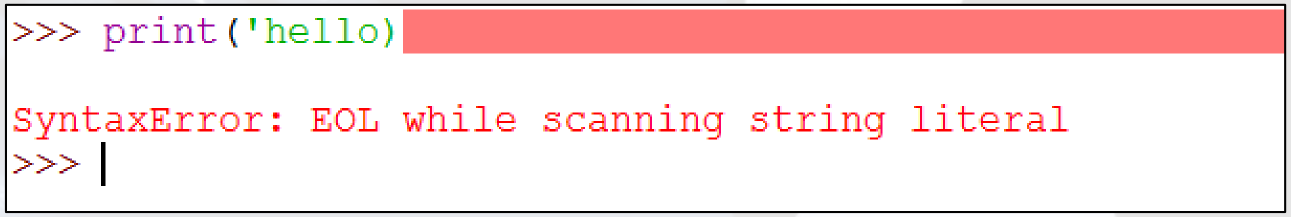
Different Types of Errors

- Different types of errors can occur – popularly known as ***bugs*** in a computer program.
  - ✓ It is important for programmers to know how to *debug* and solve those problems in the program.

- 3 types of errors
  - ✓ **Syntax errors**
  - ✓ **Runtime / Execution errors**
  - ✓ **Logic / Semantic errors**

Syntax Errors

- ❑ **Errors due to <u>violation</u> of the language syntax**
  - ✓ Python program with syntax error will not run.
  - ✓ E.g. missing end quote for string, missing: after if statement (in further topic)

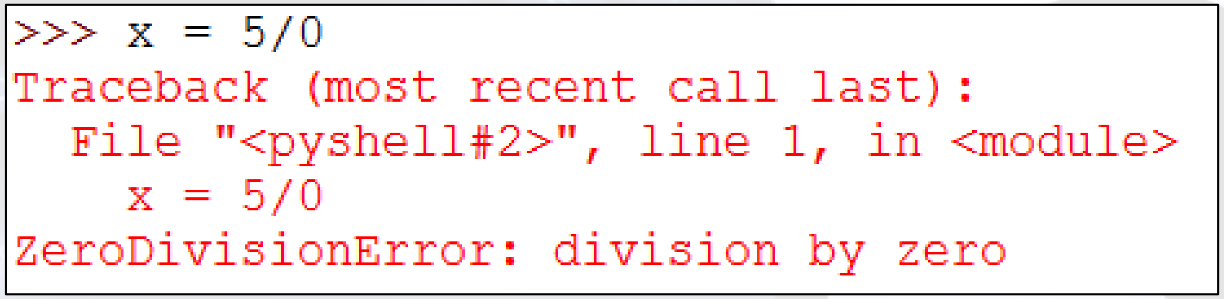


```
>>> print('hello)

SyntaxError: EOL while scanning string literal
>>> 
```

Runtime/Execution Errors

- ❑ **Errors that occur during <u>running/execution</u> of program.**
  - ✓ Python program will run until code with error, then program will terminate with error msg.



```
>>> x = 5/0
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x = 5/0
ZeroDivisionError: division by zero
```

Logic/Semantic Errors

- **Error occurs when the logic of the program is not written correctly**
  - ✓ Program is able to run but output produced is incorrect.
  - ✓ E.g. wrong formula, forgetting precedence of operators in expression, wrong condition (in further topic)

Use Debugger to solve Errors

- It is very hard to figure out the bugs in your program by eye inspection.
- Programmers usually make use of a ***Debugger***, a program that allows
  - ✓ stepping through code line by line in same order of execution
  - ✓ showing what values are stored in variables each step

Activity 3

- Go through the step-by-step guide (in Coursemology)

  Mission22-Program Errors and Debugging.pdf

  to find out more about debugger tool in IDLE.

Reading Reference

- **How to Think Like a Computer Scientist: Learning with Python 3**
  - ✓ Chapter 2

    http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html

- **PolyMall – Problem Solving and Programming**

  https://polymall.polytechnic.edu.sg/

Summary

- String Formatting
- Program Errors and Debugger