COMPSCI 101
Principles of Programming

Lecture 3: Expressions, Documentation and Modules

Learning Outcomes

At the end of this lecture, students should be able to:

- import modules and use the functions defined in the module
- use integer division and modulus operators
- include a docstring at the top of a program and use comments
- use self-documenting code to make the program easy to read and understand
- understand that an expression evaluates to one value
- understand the order of operations when an algebraic expression is evaluated
- understand how to develop a program in steps

Recap

$1 NZ = $0.95 AUS. Write a program which converts $500 NZ to Australian dollars and converts $500 AUS to New Zealand dollars using the above exchange rate. The output of the program should be:

```
amount_to_convert = 500
nz_to_aus_rate = 0.95
nz_dollars = amount_to_convert
```

NZ $500 = AUS $475.0
AUS $500 = NZ $526.3157894736842

Literals, Variables and Expressions

Literals are the actual values which can be stored in the program memory, e.g.,

- 34
- -67.5
- "a particular string"

Variables can be assigned any literal value (or an expression). Variables are used to refer to (point to) a single piece of information, e.g.,

- **result** = 567
- **final_result** = result + 45
- **phrase** = "a particular string"
- **phrase** = "Please tell me more"
- **first_name** = "Izzy"

Expressions are made up of literal values and variables. Expressions always evaluate to a single value. The right hand side of the assignment operator is an expression, e.g.,

- number = **3**
- final_result = **567 + 16 ** number**
- final_result = **final_result + number * 5 / 7**

Docstrings

A **docstring** is a special kind of string (text) used to provide documentation. A docstring:

- appears at the top of every COMPSCI 101 program,
- three double-quotes are used to surround the docstring,
- all programs should include a docstring at the top of the program,
- the docstring contains the author and a description of what the program does.

```
"""
Program which calculates the area of a circle.
Author: Damir Azhar
"""

radius = 10
area = 3.14159265359 * radius ** 2
print("Area of circle", area)
```


Comments

As well as the docstring describing the purpose of the program at the top of **ALL** our programs, comments can be added to the program code. A programming comment is a note to other programmers who need to understand the code.

- Anything between a # (hash) and the end of the line is a comment and is ignored by the interpreter

```
"""
Converts a length in inches to a length in centimetres.
Author: Damir Azhar
"""

length_in_inches = 100 #Change the value of length_in_inches here
length_in_cm = length_in_inches * 2.54

print("Length", length_in_cm)
```

```
Length 254.0
```


Use Self Documenting Code

Add comments sparingly to explain code that is difficult, or to tell other programmers something they need to know about the code.

It is always important to use good descriptive variable names.

The program below does the same job as the program on the previous slide but it uses very poor variable names which makes the program difficult to read and difficult to understand.

```
"""
Converts a length
Author: Damir Azhar
"""

a = 100
b = a * 2.54

print("Length", b)
```

```
Length 254.0
```


Skeleton of a Python Program

In general the format of a Python program is:

docstring →
initialisation →
calculation →
output →

```
"""
Calculates the area of a rectangle.
Author: Damir Azhar
"""
width = 3.56
height = 8.4

area = width * height

print("Area of rectangle", area)
```

```
Area of rectangle 29.904
```

Every Python program is stored in a file which has .py at the end of the file name (the file extension), e.g., CalculateArea.py, CompoundInterest.py

Python libraries

Python has libraries of code which contain definitions and functions which perform useful tasks and calculations. The files in these libraries are called modules. The name of a module is the name of the file without the .py extension.

The **math module** contains many useful math functions and constants, e.g., math.sin(), math.cos(), math.pow(), math.sqrt(), math.floor(), …

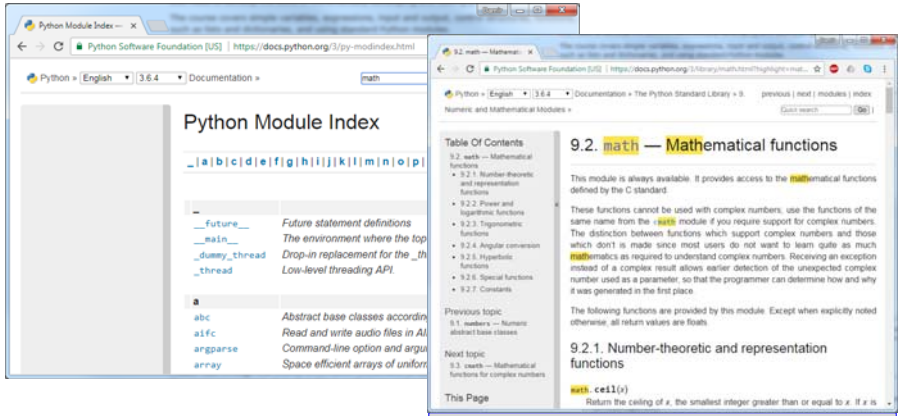
In order to be able to use the functions of a module, we need to import the module. Importing a module means that we can then use all the functions defined inside that module, e.g.,

```
"""Calculates the radius of a circle, given the area.
   Author: Damir Azhar
"""
import math
area = 221.67
radius = math.sqrt(area / math.pi)
print("Radius of circle", radius)
```

```
Radius of circle 8.399985266079987
```


www.python.org

The following website contains documentation about all the Python modules.



https://docs.python.org/3/py-modindex.html

Expressions – order of operations

Expressions containing numbers are evaluated in the same way as in mathematical expressions, i.e., BEDMAS applies:

Brackets
Exponents
Division, Multiplication
Addition, Subtraction

Note that the **/** operator always results in a float, e.g., 8 / 4 is 2.0.

- Give the output:

```
result1 = (25 - 7) * 3 + 12 / 3

result2 = 17 - 3 * 2 - 12 / 4 + 15

result3 = 32 / 4 ** (3 + 2 * 3 - 7) / 5

print(result1, result2, result3)
```

Remember to work from left to right when evaluating operators with the same priority.

More Arithmetic Operators

So far, we have seen these mathematical operators: **+, -, *, /, ****

Two more mathematical operators:

- **Floor division** (integer division) **//**
- **Modulus** (remainder) **%**

Floor division (integer division) performs the division and ignores the part after the decimal point, e.g.,

- 16 // 5 gives 3
- 17 // 5 gives 3
- 34 // 5 gives 6

Modulus performs the division and gives the remainder, e.g.,

- 16 % 5 gives 1
- 17 % 5 gives 2
- 34 % 5 gives 4
- 16 % 30 gives 16

Arithmetic Operators with Different Numeric Types

These are the mathematical operators we will be using:

+, -, *, /, **, //, %

When an arithmetic operator has operands of different numeric types, the operand with the "narrower" type is widened to that of the other operand (integer is narrower than floating point), e.g.,

- 3 % 5.0 evaluates to 3.0
- 16.0 / 8 evaluates to 2.0
- 17 // 5.0 evaluates to 3.0
- 34.0 // 5 evaluates to 6.0
- 16.0 % 5 evaluates to 1.0
- 17 % 5.0 evaluates to 2.0

Exercise

```
result1 = 25 % 3

result2 = 20 % 34

result3 = 20 // 3.0

result4 = 5 // 7

result5 = (26.7 // 1) % 3

print(result1, result2, result3, result4, result5)
```


Exercise

Order of operations:

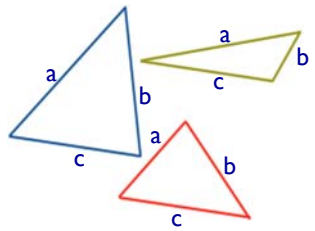
Brackets
Exponents (**)
Multiplication, Division, Modulus, Floor division
Addition, Subtraction

Give the output:

```
result1 = 25 / 4 // 3 + 4 * 10 % 3
result2 = 10 - 7 // 3 * 3 + 13 % 5 / 5 * 2
result3 = 17 % 3 * 2 - 3 ** 2 * 3 + 19 // 2
print(result1, result2, result3)
```


Heron's Formula

Heron's formula states that the area of a triangle whose sides have lengths a, b, and c is:



$$A = \frac{1}{4}\sqrt{4(a^2b^2 + a^2c^2 + b^2c^2) - (a^2 + b^2 + c^2)^2}$$

Write a program which uses **Heron's formula** to calculate and print the area of a triangle given the length of the three sides.

$$A = \frac{1}{4}\sqrt{4(a^2b^2 + a^2c^2 + b^2c^2) - (a^2 + b^2 + c^2)^2}$$

```
import math

side1 = 4
side2 = 7
side3 = 9

#Complete the code

print("Length of sides: ",side1,', ',side2,' and ',side3,sep = "")
print("Area:", area)
```

```
Length of sides: 4, 7 and 9
Area: 13.416407864998739
```


Summary

In a Python program we can:

- import modules and use the functions defined in the imported module
- use integer division and modulus operators
- use comments. Every program contains a docstring at the top of the program
- use self-documenting code to make the program easy to understand
- understand that an expression evaluates to one value
- understand the order of operations when an expression is evaluated
- understand how to develop a program in steps

Examples of Python features used in this lecture

- import modules and use the functions defined in the module

```
import math
result = math.sqrt(345)
```

- use integer division and modulus operators

```
whole_number = 456 // 3
left_overs = 456 % 12
```

- understand the order of operations when an expression is evaluated

```
result = 32 / 4 ** (1 + 2 * 3 - 7 % 4) / 5
```