CMPU4060-E
Object Oriented Software Development 1

Dr Andrew Hines

Room KE-G 026A
School of Computing
Dublin Institute of Technology

Last Week

- Course Content - description, syllabus, learning outcomes
- Assessment and Credits - how much work is there to do?
- Why do it at all? - about me and you
- Structure of the module - Lectures and Labs
- Assignments and Study - Self directed learning

Module Description

This module provides the learner with the fundamental skills of programming and object oriented programming

Module Aims

- To provide the learner with strong fundamental programming
- To provide the learner with object-oriented programming skills
- To ensure the learner has the necessary skills to design and develop an application using an object-oriented language

Learning Outcomes

On completion of this module, the student should be able to:

1. Design an object-oriented software application
2. Implement a software application using an object-oriented programming language utilising core object-oriented programming concepts, and develop problem solving skills as part of this process
3. Test and debug an object-oriented software application
4. Implement basic algorithms and data structures using an object-oriented programming language
5. Select and evaluate appropriate methods, including algorithms and patterns, for the implementation of object-oriented solutions.

Module Content

How we will deliver the Learning Outcomes?

- Fundamentals of Programming (40%)
  - Types, variables and operators
  - Control structures
  - Code style and quality
- Object Oriented Programming (40%)
  - Objects and classes
  - Methods
  - Inheritance and polymorphism
- Exception handing Data Structures and Algorithms (20%)
  - Collections
  - Basic data structures and algorithms e.g. 1D and 2D arrays, searching and sorting
  - Analysis of algorithms

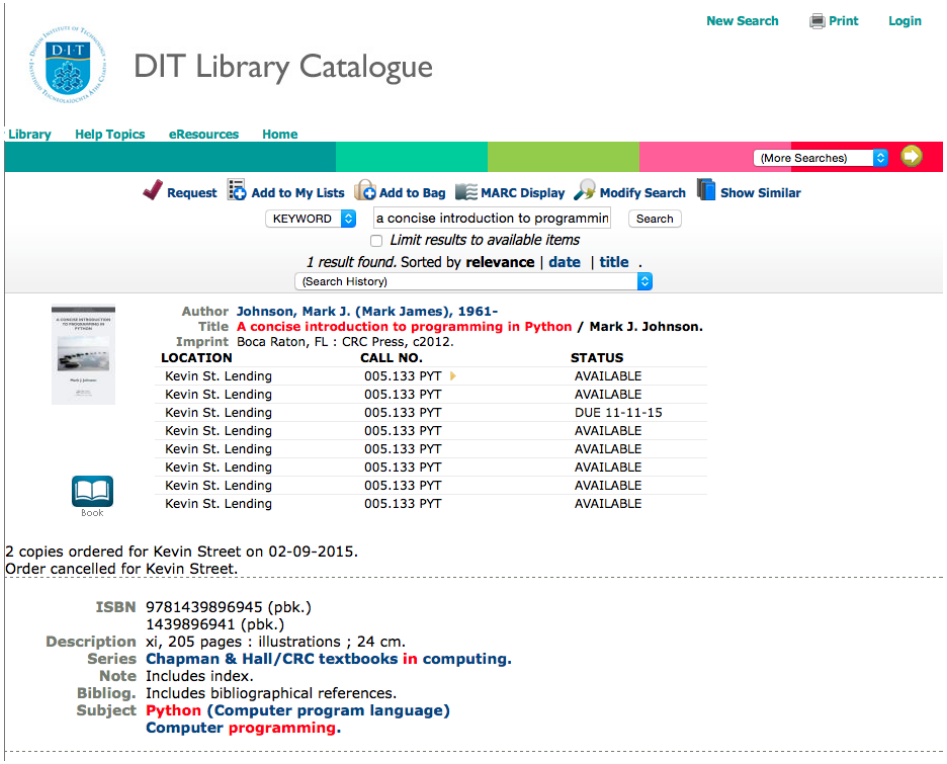
Assessment

- Continuous Assessment is carried out throughout the semester
- Plagiarism will be monitored closely in all assignments
- All assessments will be submitted via the webcourses site:
  - http://www.dit.ie/lttc/webcourseslogin/
- Additional course content will be posted on webcourses - check it regularly!

Assessment

- 1 semester= 30 ECTS credits
- 5 credits for each other core computing module
- 10 credits for this module
  - Breakdown the time:
  - At 25-30 hours effort per credit = 250 - 300 hours course work
  - 24 hours of lectures
  - 48 hours of labs = 178 hours of your own work!
  - But remember: No exam. No revision. No essays.

Textbook: Available in the Library



New Search Print Login

DIT Library Catalogue

Library Help Topics eResources Home

(More Searches)

Request Add to My Lists Add to Bag MARC Display Modify Search Show Similar

KEYWORD a concise introduction to programmin Search

Limit results to available items

1 result found. Sorted by **relevance** | **date** | **title** .

(Search History)

Author **Johnson, Mark J. (Mark James), 1961-**
Title **A concise introduction to programming in Python / Mark J. Johnson.**
Imprint Boca Raton, FL : CRC Press, c2012.

LOCATION	CALL NO.	STATUS
Kevin St. Lending	005.133 PYT	AVAILABLE
Kevin St. Lending	005.133 PYT	AVAILABLE
Kevin St. Lending	005.133 PYT	DUE 11-11-15
Kevin St. Lending	005.133 PYT	AVAILABLE
Kevin St. Lending	005.133 PYT	AVAILABLE
Kevin St. Lending	005.133 PYT	AVAILABLE
Kevin St. Lending	005.133 PYT	AVAILABLE
Kevin St. Lending	005.133 PYT	AVAILABLE

Book

2 copies ordered for Kevin Street on 02-09-2015.
Order cancelled for Kevin Street.

ISBN 9781439896945 (pbk.)
1439896941 (pbk.)
Description xi, 205 pages : illustrations ; 24 cm.
Series **Chapman & Hall/CRC textbooks in computing.**
Note Includes index.
Bibliog. Includes bibliographical references.
Subject **Python (Computer program language)**
Computer programming.

There are 10-12 copies available in Kevin St. Library

Not a mandatory text!

An excellent way to keep up and work outside of class

Online Resources

- http://learnpythonthehardway.org/book/intro.html
- https://www.codecademy.com/tracks/python
- http://www.tutorialspoint.com/python/index.htm

Computer Systems and Software

- CPU: Central Processing Unit
- RAM: Random Access Memory
- volatile: if you turn off the computer it is gone
- SSD/HDD: Solid state drive, hard disk drive
- persistent storage: saves data even if you power off the machine
- high level language: Python, C++ etc.
- instruction set: assembly language - basic commands: load data, perform arithmetic, store data
- machine language: what the CPU can processs

What happens inside a computer when your program runs?

Languages

Level	Language	Purposes
Higher	Python	Scripts
	Java	Applications
	C, C++	Applcations, Systems
	Assembly Languages	Specialized Tasks
Lower	Machine Languages	

From: Johnson, Mark J.. A Concise Introduction to Programming in Python. CRC Press, 02/2012. VitalBook file.

Compilation and Interpretation

Compiled Code

Convert the high level language version into a machine readable executable version

Interpreted Languages

Converted on the fly without an executable program

Variables, Statements and Syntax

Variables

Names that refer to data in memory - can be anything but make them sensible!* In python: A-Z, a-z, _, and 0-9**

*except for reserved *keywords* - more later **except for the first character of the variable name

Statements

A program is a sequence of *statements* that Python *interprets* and executes

Syntax

Every statement must have the correct *syntax* or form in which it is written.

Warning!

Python has different syntax in different versions

Types of Statements

The example uses 3 types of statements:

Assignment

Used to assign (give) a `variable` a value from an `expression` (left = right)

Print

Used to create output

Import

Used to access standard libraries

Data Types, Expressions and Comments

Data Types

The example contains three types: `Strings`, `Integers` and `Floats`

Expressions

Something to evaluate, e.g. input, numeric using arithmetic operations ($+$,$-$,$*$,$**$,$/$,$//$)

Comments

Comments begin with a # for single line comments. They are not interpreted - everything after the # is ignored

Another Python Program

```
# hypot.py

from math import sqrt

def myhypot(x, y):
    return sqrt(x ** 2 + y ** 2)

def main():
    a = float(input("a: "))
    b = float(input("b: "))
    print("Hypotenuse:", myhypot(a, b))

main()
```

Functions

A function Definiton

$$\texttt{def} < \texttt{function} > (< \texttt{parameters} >) :$$
$$< \texttt{body} >$$

Defining a Function

Syntax

Begin with `def` and needs a colon (:) at the end

Parameters

Parameters are used to send additional information to a function so that it can do its job. They are *optional*

Calling a Function

$$< \texttt{function} > (< \texttt{arguments} >)$$

main

`main` is a special function – called a driver. It is usually called at the bottom of the file, after everything has been defined

Function calls can be nested:

```
temp=float(input("What temperature to convert?"))
```

Return statements

`return` < `expression` >

- A return statement allows a function to output, or return data
- It is optional and the expression after is optional (causing `None` to be returned)

Local Variables

A variable with local scope exists only during the lifetime of a function execution.

- They cannot be accessed outside the function
- Their values are not remembered between function calls
- Parameter inputs are also treated as local variables

Type Conversions

Did you encounter this problem when you used the input function?

Type Conversions

Did you encounter this problem when you used the input function?

int(x)	Convert x to an interger and truncate towards 0
int(x,b)	Convert x from base b to an integer
float(x)	Convert x to a floating point
str(x)	Convert x to a string

Loops

Repetition

The ability to repeat a task over and over is a key element of programming

Maths Recap

We have some mathematicians and theoretical physicists in the class

Maths Recap

We have some mathematicians and theoretical physicists in the class

But we also have some classics and business experts

Harmonic Series

In mathematics, the harmonic series is the divergent infinite series:

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \cdots .$$

Its name derives from the concept of overtones, or harmonics in music: the wavelengths of the overtones of a vibrating string are 1/2, 1/3, 1/4, etc., of the string's fundamental wavelength. Every term of the series after the first is the harmonic mean of the neighboring terms; the phrase harmonic mean likewise derives from music.

From: https://en.wikipedia.org/wiki/Harmonic_series_(mathematics)

```
# harmonic.py

def harmonic(n):
    # Compute the sum of 1/k for k=1 to n.
    total = 0
    for k in range(1, n + 1):
        total += 1 / k
    return total

def main():
    n = int(input('Enter a positive integer: '))
    print("The sum of 1/k for k = 1 to",
        n, "is", harmonic(n))

main()
```

Ranges

```
for < variable > in < sequence >:
    < body >
```

- build-in function range – `range([start], stop, [step])`
- 0 indexing (end at index before stop)
- sequence can be a `list`

Accumulation Loops

$$< \texttt{accumulator} > = < \texttt{startingvalue} >$$

$$\texttt{loop} :$$

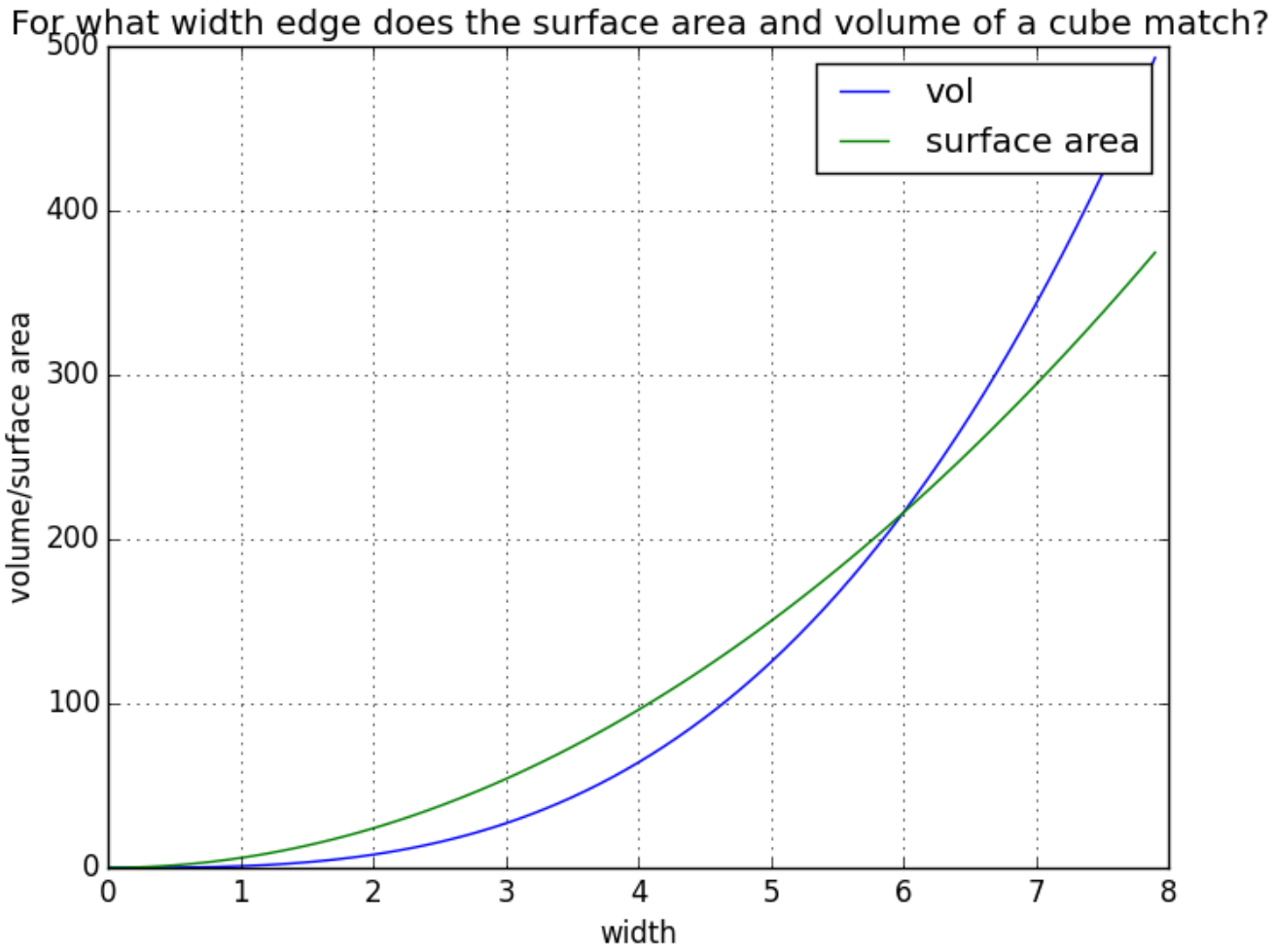
$$< \texttt{accumulator} > + = < \texttt{amounttoadd} >$$

- `+=, -=, *=, /=`
- if `x += 1` is equivalent to `x = x + 1` can you guess what the other shorthand notations mean?

Assessment Schedule

Week	Begins	Assessment	CA%
1	14-Sep		
2	21-Sep		
3	28-Sep		
4	05-Oct		
5	12-Oct	Lab/Theory Test	20%
6	19-Oct		
7	26-Oct	Review Week	
8	02-Nov		
9	09-Nov		
10	16-Nov	Lab/Theory Test	20%
11	23-Nov		
12	30-Nov		
13	07-Dec	Project Assignment	60%

Cube question from yesterday's Lab



Today's Lab: 11:00 - 13:00

- On webcourses
- You don't need to submit but you should finish the exercises section in it before the next lab