CMPU4060-E
Object Oriented Software Development 1

Dr Andrew Hines

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

Syntax and Structure: Functions and Program Flow

What exactly is a function? How should I think of it?

Lab Learning

Syntax and Structure: Functions and Program Flow

What exactly is a function? How should I think of it?

Programming Style and Quality: Ambiguity, Error Checking and Program Design

In Q5 yesterday how should I work out what the user means when they input 0.25?

Lab Learning

Syntax and Structure: Functions and Program Flow

What exactly is a function? How should I think of it?

Programming Style and Quality: Ambiguity, Error Checking and Program Design

In Q5 yesterday how should I work out what the user means when they input 0.25?

Software Design and Implementation: Actions speak louder than words

I don't need paper and a pen to write this code!

Make a Pizza

Ingredients

Dough, Sauce, Cheese, Pepperoni

Kitchen Implements and Appliances

Rolling Pin, Cheese Grater, Oven

Cooking Instuctions

Oven: 180°C, Roll Dough, add Sauce, Grated Cheese and Pepperoni, Cook for 8 minutes

Recap

Week 3

How are we doing?
25% of way through course at end of this week!

Week 1

- description, syllabus, learning outcomes
- Variables, Statements, Syntax
- Style and Quality issues: Syntax and code layout; variables and function names

Week 2

- Functions and Program Flow
- Introduced Loops and Repetition

Recap

Week 1

- description, syllabus, learning outcomes
- Variables, Statements, Syntax
- Style and Quality issues: Syntax and code layout; variables and function names

Week 2

- Functions and Program Flow
- Introduced Loops and Repetition

Week 3

- Loops and Repetition: For and While
- Data structures: Lists and Ranges

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

Question

Do you think programming is hard?

How much time did you spend working on exercises outside of the lab last week?

Question

Do you think programming is hard?

How much time did you spend working on exercises outside of the lab last week?

Double credits, double time?

Are you spending 6 hours a week working on your programming OUTSIDE of labs?

Question

Do you think programming is hard?

How much time did you spend working on exercises outside of the lab last week?

Double credits, double time?

Are you spending 6 hours a week working on your programming OUTSIDE of labs?

You *will* fall behind if you do not! – I can't put the time in for you!

What should I be doing?

Mix it up!

Any Suggestions? What have people been doing?

Suggestions?

What should I be doing?

Mix it up!

Any Suggestions? What have people been doing?

Suggestions?

- Online resources (e.g. codecademy, tutorialspoint)
- Working through exercises in the book
- Typing in example programs and seeing them work / adapting them
- Trying to solve a problem you actually want an answer to?
- Thinking of things you'd like to be able to do that we haven't covered yet (e.g. read values from a file instead of the user)

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

Do you have a better feel for what these things are after the labs?

- Choosing better variable names
- Understand the flow of a program – statements do not have to be interpreted in the order then are in the file
- Syntax – a bracket in the wrong place or a missing tab or : can break a program, even if there is no error message

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

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()
```

Sequences

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

- sequence can be a `list`
- A `list` is data structure for grouping of things, e.g. [0,1,2,3] or [10,5,1]
- build-in function range – `range([start], stop, [step])`
- 0 indexing (end at index before stop)

Lists and Ranges for Loop

Lists

A list can be any collection of things but for loops we'll consider numbers

Data structure called a `list` - it is a group of things, e.g. [0,1,2,3] or [10,5,1]

Ranges

- for simple counting there is a useful build-in function: `range`
  – `range([start], stop, [step])`
- 0 indexing (end at index before stop)

List as a loop sequence

```
def countToTen():
    listOneToTen=[1,2,3,4,5,6,7,8,9,10]

    for counter in listOneToTen:
        print("Counter is now:",counter)

def main():
    countToTen()

main()
```

Another list as a loop sequence

```
def countFromTen():
    listOddFromTenAndZero=[9,7,5,3,1,0]

    for counter in listOddFromTenAndZero:
        print("Counter is now:",counter)

def main():
    countFromTen()

main()
```

```
def manyRangeTests():

    range1=range(10)
    range2=range(1,10)
    range3=range(0,10)
    range4=range(0,10,2)
    print('--- range1 ---')
    for counter in range1:
        print(counter)
    print('--- range2 ---')
    for counter in range2:
        print(counter)
    print('--- range3 ---')
    for counter in range3:
       print(counter)
    print('--- range4 ---')
    for counter in range4:
      print(counter)

manyRangeTests()
```

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?

Does this Program make more sense now?

```
# 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()
```

Runaway or Infinite Loops

Warning

Programs with loops can get stuck running, <CTRL-C> will stop them and interrupt your program execution

What will this code do? If you have experience in C or Java the answer might surprise you...

```
for mynumber in range(10):
    print(mynumber)
    mynumber = 3
```

Uncertain Ending

Question

When we wanted to calculate the convergence of a cube's surface area and volume we could have used a for loop to try different widths

Uncertain Ending

Question

When we wanted to calculate the convergence of a cube's surface area and volume we could have used a for loop to try different widths

---

```
for sideWidth in range(0,10):
    volume = sideWidth ** 3
    surfacearea = (sideWidth ** 2) * 6 # 6 sides
    print("Width is ", sideWidth,"Volume is ",
          volume, "Surface Area is: ",
          surfacearea)
```

While Loops

Question

A while loop allows a loop to continue forever until **a condition is met**

Why might this be useful?

```
while < boolean >:
    < body >
```

Cube example with While

```
surfacearea=0
volume=-1
sideWidth=0
while surfacearea != volume:
    sideWidth+=1
    volume = sideWidth ** 3
    surfacearea = (sideWidth ** 2) * 6 # 6 sides
print("Width is ", sideWidth,"Volume is ",
      volume, "Surface Area is: ",
      surfacearea)
```

Conditional Statements

The `if` statement is a logical control statement.

`if` < `boolean` >:

    < `body` >

Example:

```
if mynumber > 0
  print(mynumber)
```

if boolean?

What is a boolean expression?

Expression	Meaning
==	equals
!=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

else

The `if` statement can be extended to say otherwise do something else:

```
if <boolean>:
  <body1>
else:
  <body2>
```

elif

...and extended again with `elif` (short for *else if*) to check for a number of conditions. Only the first true statement will execute.

```
if <boolean>:
  <body1>
elif <boolean>:
  <body2>
elif <boolean>:
  <body3>
...
else:
  <bodyN>
```

Putting it all together!

import, def, for, if, , +=, >, and a lot more!

```
# centipede.py

from turtle import *

def centipede(length, step, life):
    penup()
    theta = 0
    dtheta = 1
    for i in range(life):
        forward(step)
        left(theta)
        theta += dtheta
        stamp()
        if i > length:
            clearstamps(1)
        if theta > 10 or theta < -10:
            dtheta = -dtheta
        if ycor() > 350:
            left(30)

def main():
    setworldcoordinates(-400, -400, 400, 400)
    centipede(14, 10, 200)
    exitonclick()

main()
```

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%

Labs

Submit your code for yesterday's lab by Tuesday. (Finish Q1 – Q3)

Today's Lab

Chance to explore some code and practice functions, loops and if statements