ENGG1003 - Monday Week 2

First steps: libraries & modules, printing and plotting

Steve Weller

University of Newcastle

February 26, 2021

Lecture overview

- Python program with a library function
  - principles
  - live demo
  - LL text §1.3
- importing from modules and packages
  - principles
  - live demo
  - §1.4
- plotting, printing and input data
  - principles
  - live demo
  - §1.6
- some other stuff
  - §1.5 & §1.8

Python program with a library function

- describe the problem
- simple diagram: x, y, θ
- maybe a ball?
- algorithm is $\tan^{-1}$

The program

```
x = 10.0                    # Horizontal position
y = 10.0                    # Vertical position

angle = atan(y/x)

print((angle/pi)*180)
```

First use of a Python *function*

- first use of a *function*, in this case `atan`
- argument
- return value

Math review: radians and degrees

- Python's `atan` returns value in radians
- $\times \frac{180}{\pi}$ to get answer in degrees

Running the program

- screen grab from PyCharm – error message

Python standard library and import

- Python has plenty of functionality "built-in"
- LOTS more can be *imported*
- `atan` and other trigonometric functions not built in
- to activate that functionality, must explicitly import
- `atan` function is grouped together with many other mathematical functions in a *library module* called `math`

```
from math import atan, pi
```

The program: second attempt

```
from math import atan, pi

x = 10.0                    # Horizontal position
y = 10.0                    # Vertical position

angle = atan(y/x)

print((angle/pi)*180)
```

- script correctly produces 45.0 as output
- live demo in PyCharm shortly

Another way of importing

- use the import statement import math, but require `atan` and `pi` to be *prefixed* with math
- both techniques are commonly used and are the two basic ways of importing library code in Python

```
import math

x = 10.0                # Horizontal position
y = 10.0                # Vertical position

angle = math.atan(y/x)

print (angle/math.pi)*180
```

Live demo of Python program with a library function

blah

Importing from modules and packages

blah

Plotting, printing and input data

blah

Some other stuff

blah

Algorithms

- Informally, an *algorithm* is a series of steps which accomplishes a task
- More accurately, the steps (instructions) must:
  - Have a strict order
  - Be unambiguous
  - Be executable
- "Executable" means that the *target platform* is capable of performing that task.
  - eg: An industrial welding robot can execute "move welding tip 1 cm left". A mobile phone can't.

Algorithms

- In this course we will use:
  - *Pseudocode* to communicate algorithms to ourselves and other people
  - The Python language to communicate algorithms to computers
- Pseudocode *can* be very formal, but as engineers we will only use formal rules if required
  - eg: When documenting algorithms for other people
  - Your own "working out" can be anything that helps *you*

Algorithm Example 1

Name: Algorithm given to start my car (2015 Tarago)
Result: The vehicle's engine is idling
Initialisation: stand next to the vehicle, key fob in hand

1. Depress the unlock button on the key fob, car will beep twice
2. Place key fob in your pocket
3. Enter the vehicle, sit in the driver's seat
4. Ensure that the gear selector has P engaged
5. Depress the brake pedal
6. Press the engine start button
7. Wait 3 seconds
8. If engine is not idling
   - Call a mechanic

Example Discussion

- Algorithms typically need to feel over-explained
  - Computers are *really stupid*; get in the habit of over-thinking everything
- The algorithm contained *flow control* in the form of an "if" statement
  - The final step ("call a mechanic") was *conditional* on the car not starting

Flow Control

- Instructions in an algorithm execute in an ordered list
  - ie: top to bottom
- Flow Control is any algorithmic mechanism which changes the default "top to bottom" execution behaviour
- We will discuss IF statements
  - Another type, *loops*, discussed later
- Flow control typically requires a *condition*

Conditions

- Computers don't understand "maybe"
- A *condition* must be absolutely **true** or **false**
- Human examples:
  - I am watching a lecture
  - I am alive
  - My net worth is below AU$100M
- Computer examples:
  - `i` is less than 184
  - `x` plus `y` is not equal to zero
  - Input data has been given to the program
  - A division by zero has occurred

Code Blocks

- A *block* is a set of instructions which are grouped together
- If a single condition controls multiple instructions they can go together in a block
- In pseudocode (and Python) a block is indicated via indentation
- Eg:

```
IF it is raining
  Pack an umbrella
  Drive to campus instead of walking
  Leave home 40mins early to find parking
ENDIF
```

IF Variants

- There are several versions of IF flow control:
  - IF ... ENDIF
  - IF ... ELSE ... ENDIF
  - IF ... ELSEIF ... ENDIF
- The IF and ELSEIF keywords indicate conditions
- The ELSE keyword is *unconditional*
- Which one you choose depends on need
  - Is there one thing which is conditional?
  - Do I need to make a choice between two or more options?
  - Could nothing be executed?

IF Statement Syntax

- The IF ... ENDIF syntax is:

```
IF condition
  do some things
ENDIF
```

- Likewise: IF ... ELSEIF ... ENDIF syntax is:

```
IF condition1
  do some things
ELSEIF condition2
  do other things
ENDIF
```

- And finally:

```
IF condition
  do some things
ELSE
  do some things
ENDIF
```

IF ... ELSEIF

- The IF ... ELSEIF construct can have multiple ELSEIF sections
- A *crucial* point:
  - Conditions are only tested *if the previous ones fail*
  - Once a condition is TRUE the others are ignored
  - ie: IF - ELSE implements a choice priority

Mathematics Assumed Knowledge

- We assume you understand (and remember) up to Year 10 maths
- The course may go beyond this, but we will teach and review extra content as needed

Algorithm Example 3 - Quadratic Root Finding

From high school you should know that the equation

$$ax^2 + bx + c = 0 \tag{1}$$

has solutions given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

lets write an algorithm which provides real valued solutions to a quadratic equation.

Algorithm Example 3 - Quadratic Root Finding

Input: Real numbers a, b, and c
Output: Three numbers:

1. The number of solutions, N
2. One of the roots, x_1
3. The other root, x_2

Behaviour:

- If N is 2 then x_1 and x_2 are different real numbers
- If N is 1 then x_1 is the unique solution and x_2 is undefined
- If N is 0 then x_1 and x_2 are undefined

Algorithm Example 3 - Quadratic Root Finding

```
BEGIN
  INPUT: a, b, c
  D = b^2 - 4ac
  IF D < 0
    N = 0
  ELSEIF D == 0
    N = 1
    x1 = -b/(2a)
  ELSEIF D > 0
    N = 2
    x1 = (-b + sqrt(D))/(2a)
    x2 = (-b - sqrt(D))/(2a)
  ENDIF
END
```

- Reasonably formal pseudocode
- The IF ... ELSE IF flow control construct forces exclusive execution of only *one* block
- The first condition that is true causes execution of that block
- Subsequent blocks ignored
- Contains 3 *conditions*

Boolean Algebra Basics

- What if we want more complicated conditions? Boolean algebra is needed!
- Boolean algebra (or Boolean logic) is a field of mathematics which evaluates combinations of *logical variables* as either true or false
- Boolean *variables* can only take the values **true** (or 1) or **false** (or 0)
- Boolean algebra defines three *operators*:
  - OR
  - AND
  - NOT

Boolean Algebra Basics

- Boolean variables can be allocated any symbols (just like in "normal" algebra)
  - Typically get upper-case letters
  - eg: X = A OR B
- Various symbols can be used for OR/AND/NOT, we will only use the words here
  - Write them in capitals to remove ambiguity
  - Python uses these words in lowercase
  - Other courses (eg: ELE1710) will use different symbols again

Boolean Operators

- An *operand* is a value on which a mathematical operation takes place
  - eg: In "1 + 2" the 1 and 2 are operands and + is the operator
- OR - Evaluates true if either operand is true
  - X = A OR B
  - X is true if either one of A or B is true
- AND- Evaluates true only when *both* operands are true
  - X = A AND B
  - X is true only if both A and B are true

Boolean Operators

- OR and AND are *binary* operators
  - They operate on two operands
  - From Latin "bini" meaning "two together"
- The NOT operator is *unary*
  - It only operates on *one* operand
  - NB: The operand could be a single variable or complex expression
- NOT performs a logical inversion
  - NOT true = false
  - NOT false = true

Boolean Condition Examples

- My car needs a service if, since the last service, (more than 6 months has past) OR (more than 15000km have been travelled)
- You will pass this course if (you score 40% or more in the final exam) AND (the weighted sum of all assessments is more than 50%)
- A computer program repeats an algorithm if (there is still data to process) AND (errors have not occurred) AND (NOT (the user has terminated the program))

Reflection

- Week 1 reflection