ENGG1003 - Monday Week 4

Iteration again: `for` vs. `while` loops, debugging strategies & random numbers

Steve Weller

University of Newcastle

15 March, 2021

Lecture overview

1. iteration again (again): `for` vs. `while` loops §3.3.3
2. debugging strategies
3. random numbers in Python §2.4

1) iteration again: `for` vs. `while` loops

- side-by-side comparison for print-1-to-10
- similarities and differences
- when to use each

- XXX

Example: Finding the maximum height

- §3.3.3
- new program instead finds the maximum height achieved by the ball
- will solve in two ways: for and while

```
import numpy as np
import matplotlib.pyplot as plt

v0 = 5                          # Initial velocity
g = 9.81                        # Acceleration of gravity
t = np.linspace(0, 1, 1000)     # 1000 points in time interval
y = v0*t - 0.5*g*t**2           # Generate all heights

# At this point, the array y with all the heights is ready,
# and we need to find the largest value within y.

largest_height = y[0]           # Starting value for search
for i in range(1, len(y), 1):
    if y[i] > largest_height:
        largest_height = y[i]

print('The largest height achieved was {:g} m'.format(largest_height))

# We might also like to plot the path again just to compare
plt.plot(t,y)
plt.xlabel('Time (s)')
plt.ylabel('Height (m)')
plt.show()
```

Focus on code to find max using for

- describe strategy in words
- live demo

```
largest_height = y[0]          # Starting value for search
for i in range(1, len(y), 1):
    if y[i] > largest_height:
        largest_height = y[i]
```

```
The largest height achieved was 1.27421 m
```

Focus on code to find max using while

- describe strategy in words
- live demo

```
i = 0
while y[i+1] > y[i]:
    i = i + 1
```

2) debugging strategies

1. running code by hand
2. don't guess, print!
3. take baby steps
4. use comments to hide debug code

running code by hand

- Work through your code line-by-line, with a piece of paper and a pen
- Use paper/notes-app before you run code, so that you know what you want program to calculate on each line before you run code
- Second-best: use notes app on iPad or similar—idea is to *think before computing*
- Check that every line of code, with every use of an array index (hint) matches what you expect
- Near enough isn't good enough here. Think like a computer: work systematically through each line of code. Is it doing what you want it to do?
- It's easy (but often misleading) to look at code and

don't guess, print!

- xxx

take baby steps

- xxx

use comments to hide debug code

- but don't delete it

3) random numbers in Python

- Python provides ability to produce (apparently) random numbers
- referred to as *pseudo-random numbers*
- these numbers are not truly random, but produced in a predictable way once a *seed* has been set
- the seed is a number which depends on the current time

Drawing **one** random number at a time

- LL p54
- `throw_2_dice.py`
- live demo
- function randint is available from the imported module random, which is part of the standard Python library
- randint(a,b) returns a pseudo-random *integer* in the range $[a, b]$ where $a \leq b$

Fixing the seed

- When debugging programs that involve pseudo-random number generation, it is a great advantage to *fix the seed*
- ensures that the very same sequence of numbers will be generated each time the code is run
- simply means that you pick the seed yourself and tell Python what that seed should be
- example: random.seed(10)
- live demo of `throw_2_dice.py`

Two useful functions: random and uniform

- both of these functions return a floating point number from an interval where each number has equal probability of being drawn
- confusing: random function in random module
- random, the interval is always [0, 1) (i.e. 0 is included, but 1 is not)
- uniform requires the programmer to specify the interval [a, b] (where both a and b are included)
- text doesn't mention Gaussian (normal) random numbers

Live demo: random and uniform

- LLp55 screengrab
- live demo
- a word about In[] and Out[]

```
In [1]: import random

In [2]: x = random.random()          # draw float from [0, 1), assign to x

In [3]: y = random.uniform(10, 20) # ...float from [10, 20], assign to y

In [4]: print('x = {:g}, y = {:g}'.format(x, y))
Out[5]: x = 0.714621 , y = 13.1233
```

Drawing **many** random numbers at a time

- three PRN generators so far: one at a time
- could use in a loop, call once each time
- much better: vectorization (have seen before)
- need another module called random
  - this one inside numpy library

Live demo: random numbers from numpy library

- LLp55b screen grab
- live demo
- np.random.randint
  - np library, random module, randint function

```
In [1]: import numpy as np

In [2]: np.random.randint(1, 6, 4)      # ...4 integers from [1, 6)
Out[2]: array([1, 3, 5, 3])

In [3]: np.random.random(4)          # ...4 floats from [0, 1)
Out[3]: array([ 0.79183276,  0.01398365,  0.04982849,  0.11630963])

In [4]: np.random.uniform(10, 20, 4)    # ...4 floats from [10, 20)
Out[4]: array([ 10.95846078,  17.3971301 ,  19.73964488,  18.14332234])
```

- live demo
- numpy allows the seed to be set. For example, setting the seed to 10 (as above), could be done by np.random.seed(10)

Lecture summary

- Iteration again
  - for vs while
- debugging strategies
  - blah
  - blah
  - blah
- random numbers
  - random module: randint, random and uniform
  - random module in numpy library: randint, random and uniform