LECTURE 2
STRINGS AND LISTS

Fundamentals of Programming - COMP1005

Semester 1, 2017

Department of Computing

Curtin University

Copyright Warning

Learning Outcomes

- Define and use more complex datatypes (strings and lists) and variations on control structures
- Use slicing and indexing to access elements in a list
- Use a supplied Python package to provide random number options
- Understand and implement simple Monte Carlo algorithms

STRINGS

Fundamentals of Programming

Lecture 2

Strings

- We have already seen strings in many of the previous examples:
  - print(‘TICKET MACHINE’)
- Strings are a sequence of characters
- Some characters are special - \n is a new line
  - print(‘\nTICKET MACHINE\n’)
- Characters are alphabetical (upper and lower), numbers, symbols and spaces
- The order of the characters matters, so a string is referred to as a **sequence**

Defining strings

- Quotes indicate 42 is a string, not a number
  - `'42'` `'42'` is the character '4', followed by '2'
    `int('42')` is a number (101010_2)

- A string is defined using matching quotation marks
  - `"String 1"`
  - `'String 2'`
- If we mix the quotations marks, we get a syntax error

```
$ python stringex.py
  File "stringex.py", line 3
    string3 = 'String 3"
                        ^

SyntaxError: EOL while scanning string literal
```

Escaping characters

- If we need our string to contain a quotation mark or an apostrophe, we can “**escape”** it with a “\”
  - `grail = 'It\'s just a flesh wound'`
  - `brian = 'Now you listen here! He\'s not the Messiah. He\'s a very naughty boy!'`
- Or we can use double quotes outside and singles inside:
  - ```
    walks = "I'm sorry to have kept you waiting, " \
            "but I'm afraid my walk has become " \
            "rather sillier lately.”
    ```
  - `print(Grail + "\n" + Brian + '\n' + Walks)`
- **Note**: MS Powerpoint and Word will change the quote characters to “” ‘’ instead of `""''` – these will not be recognised by Python
    ```

Special characters

- A few times we've used \n to insert a new line into a string
- Newlines are not the only special characters we might want to use
- Tabs \t can be useful for formatting columns
- Also \b backspace, \f formfeed, \r carriage return
- If we want to print "\n", we need to escape the escape character:
  - `print('Use \\n for newline:\n…')`
  - `print('Use \\t for tab:\t!')`

Escape Characters

```
print("spam\nspam\tspam"
      "\fspam\bspam\rSPAM")
```

```
spam\n
spam\t spam\f   \b
SPAM            spaspam\r
```

Long strings

- Python style suggests limiting each line of code to **79** characters
- This lets you have multiple windows open at the same time with every line fully visible

- To have a long string across multiple lines, split the string into smaller strings and add a \
  - ```
    walks = "I'm sorry to have kept you waiting, " \
            "but I'm afraid my walk has become " \
            "rather sillier lately.”
    ```
  - *Style guide advises to line up the opening quotes on each line*

- If you're inside brackets, no need for \
  - ```
    print("Now you listen here! He's not the Messiah. "
          "He\'s a very naughty boy!")
    ```

Very long strings

- Use triple quotes to create a very long string, wrapping across multiple lines:

```
parrot = """This parrot is no more. It has
ceased to be. It's expired and gone to meet
its maker. \nThis is a late parrot. \nIt's
a stiff. Bereft of life, it rests in peace.
If you hadn't nailed it to the perch, it
would be pushing up the daisies. It's rung
down the curtain and joined the choir
invisible. \nThis is an ex-parrot."""
print(parrot)
```

- There is no need to escape the apostrophies within triple quotes – much easier to maintain

Length of strings

- Every string has a function len() to get the string length
- The len() function counts all the characters, including spaces, newlines and tabs, to get the length

```
ni = 'Ni!'
print('Length of string is: ', len(ni))
```

```
Length of string is: 3
```

```
print('Length of string is: ', len(parrot))
```

```
Length of string is: 315
```

STYLE

Fundamentals of Programming

Lecture 2

Python Style

- Python is a community development, with "**Python Enhancement Proposals**" or "PEP"s used to define and pitch for changes/standards
- PEP-8 provides a style guide, which we will be using in this unit

  https://www.python.org/dev/peps/pep-0008/
- These are not **rules**, but **guidelines** to help with consistency and readability
- Guido says:

  "Code is read much more often than it is written".
- And PEP-20 (Zen of Python) says:

  "Readability counts."

PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Contents

Style in this unit

- You will need to read and modify your code, and we will need to read and assess your code
- Readability counts
- Follow PEP-8 throughout this unit
- We will write a README file for each practical, test and for the assignment
- We will also require comments at the start of each program, e.g.

```
#
# Author    : Michael Palin
# ID        : 12345678
#
# numbers2.py -  Read in a list of numbers (negative to
#                exit) and give the sum of the numbers
#
# Revisions: 8/3/2017 – fixed style to comply with PEP-8
#          : 2/3/2017 – created
#
```

Style beyond this unit

- After this unit, you may be part of a project that uses a different style.
- When in Rome, do as the Romans do…
  ...follow the project style.
- **PEP-8**:
  "A style guide is about consistency.
  Consistency with this **style guide** is important.
  Consistency within a **project** is more important.
  Consistency within one **module or function** is the most important."
- When writing your own code, you may have to define your own style – PEP-8 is an excellent starting point

INDEXING

Fundamentals of Programming

Lecture 2

Indexing

- As a sequence, we can assign a number to each element in a string:

```
witches = 'Now, why do witches burn?'
```

N	o	w	,		w	h	y		d	o		w	i	t	c	h	e	s		b	u	r	n	?
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

- Counting starts at zero
- Escaped characters only count as one char

```
blackknight = 'It\'s just a flesh wound.'
```

I	t	'	s		j	u	s	t		a		f	l	e	s	h		w	o	u	n	d	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Accessing individual characters

- Once we have numbers assigned to each character position, we can pick out individual characters
- Element zero is the first letter "`I`"
  - `blackknight[0]` is "`I`"
  - `blackknight[2]` is "`'`"
  - `blackknight[11]` is "` `"
  - `blackknight[23]` is "`.`"

I	t	'	s		j	u	s	t		a		f	l	e	s	h		w	o	u	n	d	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Accessing individual characters

- Using the indexes, we can access characters within a string
- Putting this together with the string length, we can loop through the characters

```
for index in range(len(blackknight)):
    print(blackknight[index])
```

```
I
t
'
s

j
u
s
t
```

len(blackknight) is 24

range(24) goes from 0 to 23

Working with range() - subrange

- Range can give a sequence of numbers in a **range** going forward, backward or skipping…
- range([start] ,stop, [step])

```
for index in range(12, len(blackknight)):
    print(blackknight[index])
f
l
e
s
h

w
o
u
n
d
!
```

len(blackknight) is 24

range(12,24) goes from 12 to 23

Working with range() - reverse

- Range can give a sequence of numbers in a range going forward, **backward** or skipping…
- range([start] ,stop, [step])

```
for index in range(len(blackknight)-1, -1, -1):
    print(blackknight[index])
```

```
!
d
n
u
o
w

h
s
```

len(blackknight-1) is 23

range(23,-1,-1) goes from 23 down to 0

Working with range() - skip

- Range can give a sequence of numbers in a range going forward, backward or **skipping**…
- range([start] ,stop, [step])

```
for index in range(0, len(blackknight), 2):
    print(blackknight[index])
```

```
I
'

u
t
a
f
e
```

len(blackknight) is 24

range(0,24,2) goes from 0 to 23 in 2's: *0,2,4,6,8...22*

Working with range() – reverse & skip

- Range can give a sequence of numbers in a range going forward, **backward** or **skipping**…
- range([start] ,stop, [step])

```
for index in range(len(blackknight)-1, -1, -3):
    print(blackknight[index])
```

```
!
u

e

t
j
'
```

len(blackknight-1) is 23

range(23,-1,-3) goes from 23 down to 0 in 3's: *23,20,17...*

Using negative numbers

- Sometimes it's useful to work back from the end of the string
- This is done using negative numbers
- Element 10 is 'e' and is also element -1
  - `johncleese[-1]` is "e"
  - `johncleese[-7]` is " "
  - `johncleese[-11]` is "J"

J	o	h	n		C	l	e	e	s	e
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Another example…

- In blackknight, element 23 is '.' and is also element -1
  - `blackknight[-1]` is "."
  - `blackknight[-5]` is "'"
  - `blackknight[11]` is " "
  - `blackknight[23]` is "."

I	t	'	s		j	u	s	t		a		f	l	e	s	h		w	o	u	n	d	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

WORKING WITH STRINGS

Building and Operating on Strings

Building strings

- The main operator for string expression is "+" or concatenate
- Concatenate adds the strings together one after the other – no spaces

```
name = 'John' + 'Cleese'
```

- …will give us 'JohnCleese'

```
name = 'John' + ' ' + 'Cleese'
```

- …will give 'John Cleese'

Printing strings - separators

- When printing strings we have more flexibility than concatenating
- If we want a character printed between each variable, we use **sep=**
- Default separator is ' '

```
print(eric, graham, terry, sep='*')
Eric Idle*Graham Chapman*Terry Gilliam

print(eric, graham, terry, sep='')
Eric IdleGraham ChapmanTerry Gilliam

print(eric, graham, terry, sep=',')
Eric Idle,Graham Chapman,Terry Gilliam
```

Printing strings - ends

- If we want a character printed at the end of each line, we use **end=**
- Default separator is '\n'
- Handy for keeping lines together when printing in loops

```
for index in range(len(blackknight)-1, -1, -1):
    print(blackknight[index], end=' ')
! d n u o w   h s e l f   a   t s u j   s ' t I

for index in range(len(blackknight)-1, -1, -1):
    print(blackknight[index], end='')
!dnuow hself a tsuj s'tI
```

Working with strings

Operation	Result
x in s	True if an item of s is equal to x, else False
x not in s	False if an item of s is equal to x, else True
s + t	the concatenation of s and t
s * n or n * s	equivalent to adding s to itself n times
len(s)	length of s
min(s)	smallest item of s
max(s)	largest item of s
s.index(x[, i[, j]])	index of the first occurrence of x in s (at or after index i and before index j)
s.count(x)	total number of occurrences of x in s

Working with strings - examples

```
menuitem1 = 'Spam, egg, spam, spam, bacon and spam'
spam = 'spam, '
menuitem2 = 'Spam, ' + spam*6 + 'baked beans, ' + spam*2 +
'spam and spam'
print("Min value: ", min(menuitem1))
print("Max value: ", max(menuitem1))
print(menuitem2)
print("Spam count: ", menuitem1.count('spam'))
print("Comma count: ", menuitem1.count(','))
print("Spam at: ", menuitem2.index('spam'))
print("Spam at: ", menuitem2.index('spam',10, 20)
```

Working with strings - examples

```
menuitem1 = 'Spam, egg, spam, spam, bacon and spam'
spam = 'spam, '
menuitem2 = 'Spam, ' + spam*6 + 'baked beans, ' + spam*2 +
'spam and spam'
print("Min value: ", min(menuitem1))
print("Max value: ", max(menuitem1))
print(menuitem2)
print("Spam count: ", menuitem1.count('spam'))
print("Comma count: ", menuitem1.count(','))
print("Spam at: ", menuitem2.index('spam'))
print("Spam at: ", menuitem2.index('spam',10, 20)
```

```
Min value:
Max value:  s
Spam, spam, spam, spam, spam, spam, spam, baked beans, spam,
spam, spam and spam
Spam count:  10
Comma count:  10
Spam at:  6
Spam at:  12
```

Unicode chart – basic Latin

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077

8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

String methods

Method	Result
s.upper()	Returns a copy of s with all elements converted to uppercase
s.lower()	Returns a copy of s with all elements converted to lowercase
s.startswith(pre)	Returns True if s starts with pre
s.endswith(post)	Returns True if s ends with post
s.replace(old, new[, count])	Returns a copy of the string with [the first count] occurrences of old replaced with new
s.strip()	Return a copy of the string with leading and trailing whitespace removed (spaces, tabs etc)
s.isnumeric()	Return True if string has only numeric chars

Working with strings - examples

```
spam = 'Spam'
print(spam.upper())
print(spam.lower())
if spam.startswith('Sp'):
    print(spam + ' Starts with: ' + 'Sp')
print(menuitem2.replace('spam','egg'))
```

```
SPAM
spam
Spam Starts with: Sp
Spam, egg, egg, egg, egg, egg, egg, baked beans,
egg, egg, egg and egg
```

•

LISTS

Fundamentals of Programming

Lecture 2

Lists

- If you need to keep lots of data in one place, then you can put it into a list
- Lists can contain numbers, strings, other lists, or a combination
- Items in a list are kept in order
- You can access elements with an index (like we saw with strings)
- You can change, delete, or add to the items in a list at any point

Lists example

```
pythons = ['John','Michael','Terry','Graham','Eric']
print(pythons[1])                    #is Michael
pythons[1] = 'Michael Palin'  #updates the value 1

del pythons[3]                       #deletes 'Graham'
print(pythons[3])                    #is Eric

pythons.append('Graham')       #adds Graham in pos 4
pythons.insert(2, 'Douglas Adams') #honorary python?

pythons.remove('Douglas Adams')  # perhaps not
```

Using lists

```
pythons = ['John', 'Michael', 'Terry', 'Graham', 'Eric']
for monty in pythons:
    print('Legend: ', monty)
```

```
Legend:  John
Legend:  Michael
Legend:  Terry
Legend:  Graham
Legend:  Eric
```

Lists are **sequences**,
they have **order**,
so can be used on **for loops**

```
x = [1, 2, 3, 'four']
y = [5, 6, 7, 8.0]
z = x + y
print(z)
```

Lists can hold varied types of data – ints, strings, float, even other lists!

```
[1, 2, 3, 'four', 5, 6, 7, 8.0]
```

Lists within lists

```
menu = [['egg', 'bacon'],
        ['egg', 'sausage', 'bacon'],
        ['egg', 'spam'],
        ['egg', 'bacon', 'spam'],
        ['egg', 'bacon', 'sausage', 'spam'],
        ['spam', 'bacon', 'sausage', 'spam'],
        ['spam', 'egg', 'spam', 'spam', 'bacon', 'spam'],
        ['spam', 'sausage', 'spam', 'spam', 'bacon', \
        'spam', 'tomato', 'spam']]


print(menu[4])
['egg', 'bacon', 'sausage', 'spam']

print(menu[4][2])
sausage
```

From strings to lists (of strings)

- A common and regular task is to split a strings into pieces, based on a delimiter.
- The split method returns a list of strings
- Makes easy work of handling comma separated files

```
menuitem2 = 'Spam, ' + spam*6 + 'baked
beans, ' + spam*2 + 'spam and spam'
ingredients = menuitem2.split(',')
print(ingredients)
```

```
['Spam', ' spam', ' spam', ' spam', ' spam',
' spam', ' spam', ' baked beans', ' spam', '
spam', ' spam and spam']
```

SLICING

Fundamentals of Programming

Lecture 2

Slicing

- We can slice strings and lists to access parts of them.
- Similar to how we could use start, stop and step with the range function…
- `mystring[[start]: [stop]: [step]]`

```
name = 'John' + ' ' + 'Cleese'
name[5:10] => 'Clees'
```

- `If any are omitted, they default to 0, size and 1 respectively`

```
name[:4] => 'John'
name[4:] => ' Cleese'
```

Slicing – step and negative

```
name = 'John' + ' ' + 'Cleese'
name[:-2] => 'John Clee'
name[-6:-2] => 'Clee'

name[0:-1:2] => 'Jh le'
name[-1:0:-2] => 'eel h'
name[:4:3] => 'Jn'
name[4::3] => ' ee'
```

Indices

- One way to remember how slices work is to think of the indices as pointing *between* characters, with the left edge of the first character numbered 0.

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1   .
```

- The first row of numbers gives the position of the indices 0...6 in the string; the second row gives the negative indices.
- The slice from *i* to *j* consists of all characters between the edges labeled *i* and *j*, respectively.
- For non-negative indices, the length of a slice is the difference of the indices

(PSEUDO) RANDOM NUMBERS

Fundamentals of Programming

Lecture 2

Uses material from https://pymotw.com/3/random/index.html

Generating random numbers

- The **`random` module** provides random number generation for python
- Calling the `random()` function returns the next random floating point value from the generated sequence
- All of the returned values fall within the range 0 <= n < 1.0

```
import random

for i in range(5):
   print(random.random(), end=‘ ‘)
print()
```

```
0.9017800331429163 0.1327143209055359
0.5686552453817835 0.0752634349980656
0.546624554059005
```

Seeding

- random() produces different values each time it is called
- There is a long period before it repeats any numbers
- If you want to be able to repeat your experiment, you can use a seed value
- The same values will come up every time you run the code

```
import random

random.seed(1)
for i in range(5):
   print(random.random(), end=‘ ‘)
print()
```

```
0.13436424411240122 0.8474337369372327
0.763774618976614 0.2550690257394217
0.49543508709194095
```

Random integers

- random() generates floating point numbers.
- The best way to generate integers is with randint()
- The arguments to randint() are the inclusive range for the values:

```
print(random.randint(1,100), end=' ')
```

- randrange() gives the option for a step argument (start, stop, step):

```
print(random.randrange(0, 101, 5)
```

Picking random items

- The choice() function makes a random selection from a sequence
- In this case, the sequence is 0 and 1 representing heads or tails
- Some of the code will be unfamiliar, so just gloss over it

```
import random

sides = [0, 1]
heads = 0
tails = 0

for i in range(1000):
    if (random.choice(sides) == 0):
        heads += 1
    else:
        tails += 1

print("Heads = ", heads)
print("Tails = ", tails)
```

Picking random items

- The choice() function makes a random selection from a sequence
- In this case, the sequence is 0 and 1 representing heads or tails

```
import random

sides = [0, 1]
heads = 0
tails = 0

for i in range(1000):
    if (random.choice(sides) == 0):
        heads += 1
    else:
        tails += 1

print("Heads = ", heads)
print("Tails = ", tails)
```

SAMPLE OUTPUT

```
$ python choices.py
Heads =  515
Tails =  485
$ python choices.py
Heads =  488
Tails =  512
$ python choices.py
Heads =  506
Tails =  494
```

In addition

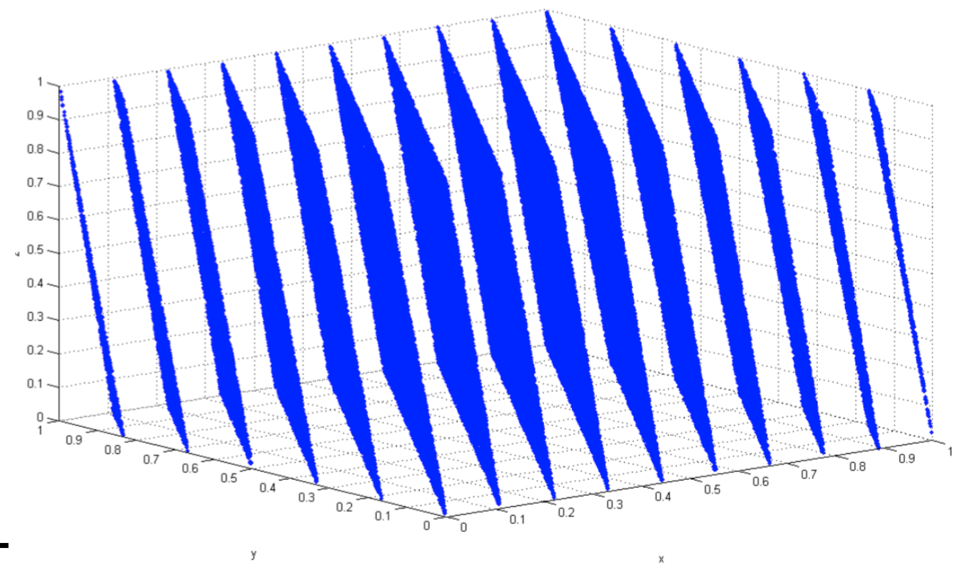
- The `random` module supports
  - Saving state
  - Permutations
  - Sampling
  - Multiple simultaneous generators
  - Non-uniform distributions

About Random Number Generation

- No such thing as random number generation – proper term is pseudorandom number generator (PRNG)
- Generate long sequence of numbers that seems “random”
- Properties of good PRNG:
  - Very long period
  - Uniformly distributed
  - Reproducible
  - Quick and easy to compute

Pseudorandom Number Generator

- Generator from lcgenerator.h is a Linear Congruential Generator (LCG)
  - Short period (= PMOD, 714025)
  - Not uniformly distributed – known to have correlations
  - Reproducible
  - Quick and easy to compute
  - Poor quality (don’t do this at home)



Correlation of RANDU LCG (source: http://upload.wikimedia.org/wikipedia/commons/3/38/Randu.png)

Good PRNGs

- For serial codes
  - Mersenne twister – **used in Python**
  - GSL (GNU Scientific Library), many generators available (including Mersenne twister)
  - http://www.gnu.org/software/gsl/
- For parallel codes
  - SPRNG, regarded as leading parallel pseudorandom number generator
  - http://sprng.cs.fsu.edu/

RANDOM.ORG

- Offers *true* random numbers to anyone on the Internet.
- The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs.
- People use RANDOM.ORG for holding drawings, lotteries and sweepstakes, to drive online games, for scientific applications and for art and music.
- The service has existed since 1998 and was built by Dr Mads Haahr of the School of Computer Science and Statistics at Trinity College, Dublin in Ireland.

MONTE CARLO TECHNIQUES

Fundamentals of Programming

Lecture 2

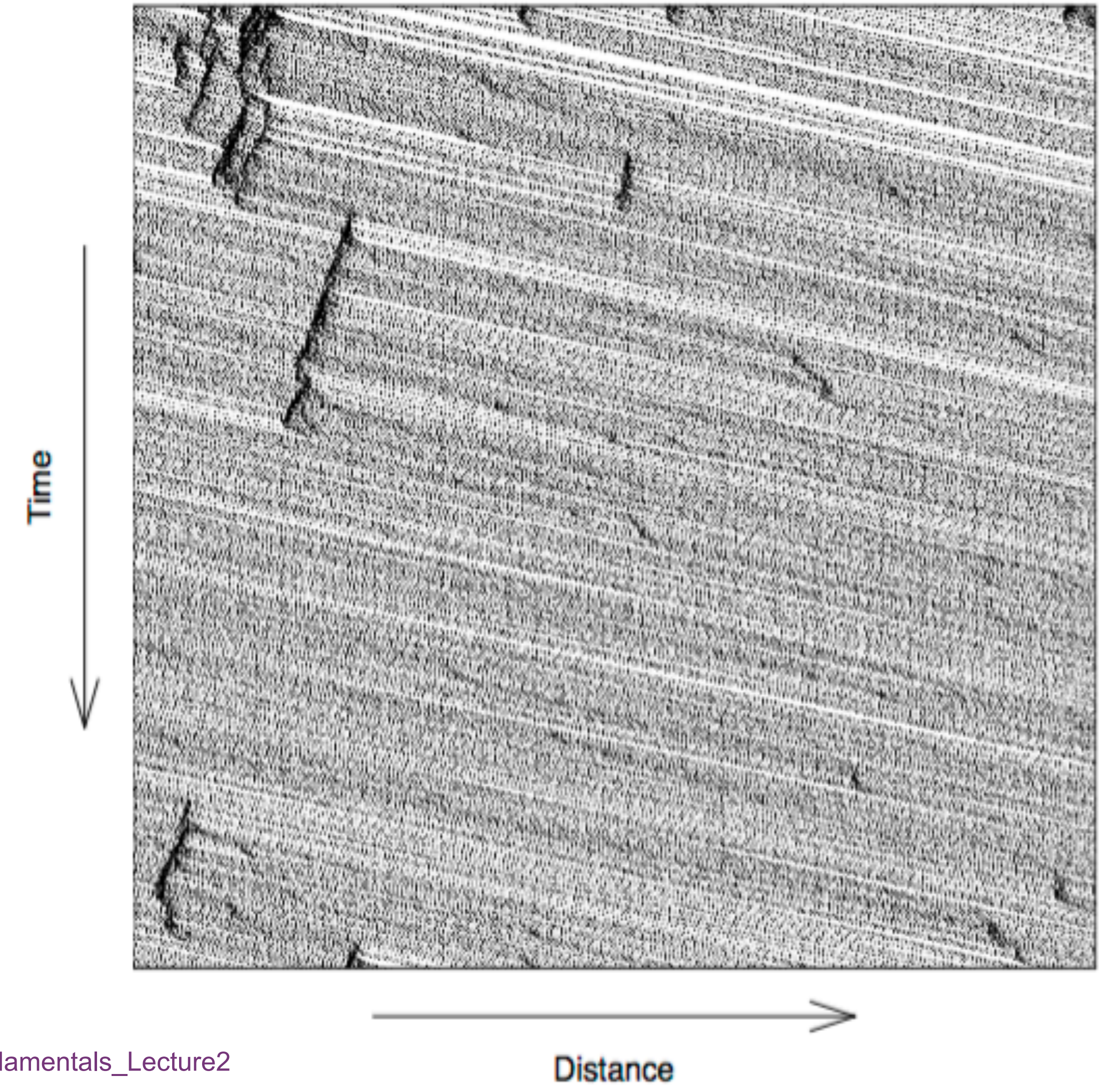
Monte Carlo Techniques

- The core idea of Monte Carlo is to learn about a system by simulating it with random sampling
- It is powerful, flexible and very direct
- MC is often the simplest way to solve a problem, and sometimes the only feasible way
- The Monte Carlo method is used in almost every quantitative subject of study:
  - physical sciences, engineering, statistics, finance, and computing, including machine learning and graphics

Example: Traffic Modelling

- We can model the occurrence of traffic jams
- At places where the number of traffic lanes is reduced, cars slow down and form a blockage
- Similarly, accidents or poor visibility or the occasional slow vehicle can bring about a traffic jam
- Sometimes a traffic jam spontaneously appears in flowing traffic, and moves slowly backwards against the traffic
- We can model this with Monte Carlo techniques

Nagel-Schreckenberg model



100 cars in simulation, each simulatenously evaluating four rules:

$v \leftarrow \min(v + 1, v_{max})$

$v \leftarrow \min(v, d - 1)$

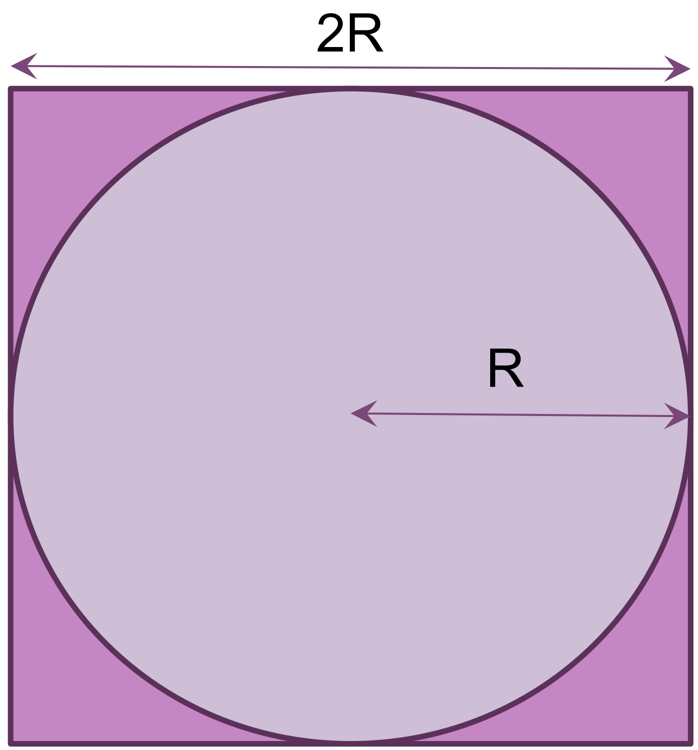
$v \leftarrow \max(0, v - 1)$ with probability p

$x \leftarrow x + v$

(speed, distance, slow down, new position)

http://statweb.stanford.edu/~owen/mc/

Method of Darts



- Imagine a dartboard with a circle of radius R inside a square
- Area of circle = πR^2
- Area of square $(2R)^2 = 4R^2$

$$\frac{\text{Area of circle}}{\text{Area of square}} \quad \frac{\pi R^2}{4R^2} \quad \frac{\pi}{4}$$

Ratio of areas is proportional to π

How to find area?

- Suppose we threw darts (completely randomly) at the dartboard
- Count # darts landing in circle & total # darts landing in square
- Ratio of these numbers gives approximation to ratio of areas
- Quality of approximation increases with # darts thrown

Method of Darts

- $\pi = 4 \times \frac{\#\text{darts inside circle}}{\#\text{ darts thrown}}$
- How in the world do we simulate this experiment on a computer?
  - Decide on length R
  - Generate pairs of random numbers (x, y) s.t. -R ≤ (x, y) ≤ R
  - If (x, y) within circle (i.e., if (x2+y2) ≤ R2) add one to tally for inside circle
  - Lastly, find ratio
- *Note: this is a highly inefficient approach for calculating pi*

Code

```
import random
num_trials = 1000000

ncirc = 0
r = 1.0       # radius of circle
r2 = r*r

for i in range(num_trials):
    x = random.random();
    y = random.random();
    if ((x*x + y*y) <= r2):
        ncirc += 1

pi = 4.0 * ncirc / num_trials

print("\nFor ", num_trials, " trials, pi = ", pi)
```

```
For  1000000  trials, pi =  3.141388
```

A more complex function to evaluate

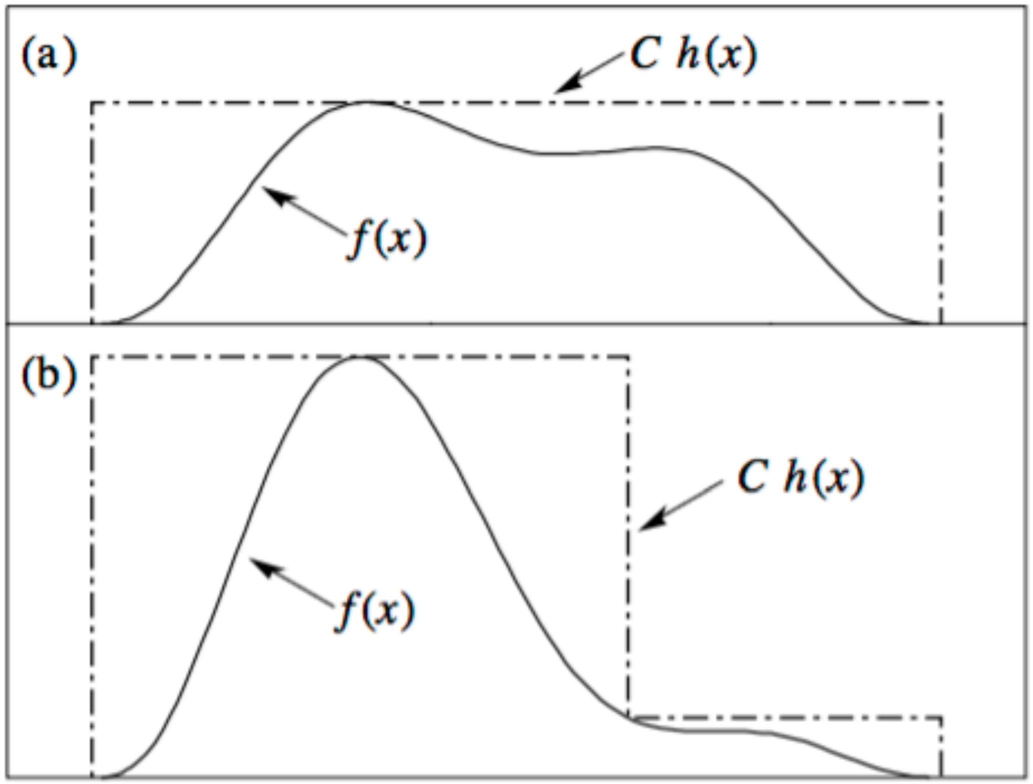


Figure 33.2: Illustration of the acceptance-rejection method. Random points are chosen inside the upper bounding figure, and rejected if the ordinate exceeds $f(x)$. The lower figure illustrates a method to increase the efficiency (see text).

http://pdg.lbl.gov/2009/reviews/rpp2009-rev-monte-carlo-techniques.pdf

Summary

- We’ve discussed the use of strings and lists
- We’ve shown how to use slicing and indexing to access elements in a list
- We’ve shown how sequence types can be used in loops
- We know how to generate pseudo-random numbers
- We have seen some Monte Carlo algorithms
- All of which will be applied in the practicals

PRACTICAL SESSIONS

Practical 1

- Covered a lot of ground – but almost everyone finished it within the two hours.
- Make sure you understand it before moving on to Prac 2
- If you haven’t finished, you are welcome to come to additional pracs
- This unit is not a competition – we don’t scale…
- … so you may as well help each other!
- (Just not on the assignment!)

Practical Sessions - Review

```
#
# growth.py - simulation of unconstrained growth
#
print("\nSIMULATION - Unconstrained Growth\n")
length = 10
population = 100
growth_rate = 0.1
time_step = 0.5
num_iter = length / time_step
growth_step = growth_rate * time_step
print("INITIAL VALUES:\n")
print("Simulation Length (hours): ", length)
print("Initial Population: ", population)
print("Growth Rate (per hour): ", growth_rate)
print("Time Step (part hour per step): ", time_step)
print("Num iterations (sim length * time step per hour): ", num_iter)
print("Growth step (growth rate per time step): ", growth_step)
```

Practical Sessions - Review

```
print("\nRESULTS:\n")
print("Time: ", 0, " \tGrowth: ", 0, " \tPopulation: ", 100)
for i in range(1, int(num_iter) + 1 ):
    growth = growth_step * population
    population = population + growth
    time = i * time_step
    print("Time: ", time, " \tGrowth: ", growth,
          " \tPopulation: ", population)
print("\nPROCESSING COMPLETE.\n")
```

```
SIMULATION - Unconstrained Growth

INITIAL VALUES:

Simulation Length (hours):  10
Initial Population:  100
Growth Rate (per hour):  0.1
Time Step (part hour per step):  0.5
Num iterations (sim length * time step per hour):  20.0
Growth step (growth rate per time step):  0.05
```

Practical Sessions - Review

```
RESULTS:

Time:  0        Growth:  0                   Population:  100
Time:  0.5      Growth:  5.0                 Population:  105.0
Time:  1.0      Growth:  5.25                Population:  110.25
Time:  1.5      Growth:  5.5125              Population:  115.7625
Time:  2.0      Growth:  5.788125000000001   Population:  121.550625
Time:  2.5      Growth:  6.07753125          Population:  127.62815624999999
Time:  3.0      Growth:  6.3814078125        Population:  134.00956406249998
Time:  3.5      Growth:  6.700478203124999   Population:  140.71004226562496
Time:  4.0      Growth:  7.0355021132812485  Population:  147.7455443789062
Time:  4.5      Growth:  7.387277218945311   Population:  155.13282159785152
Time:  5.0      Growth:  7.756641079892576   Population:  162.8894626777441
Time:  5.5      Growth:  8.144473133887205   Population:  171.0339358116313
Time:  6.0      Growth:  8.551696790581564   Population:  179.58563260221285
Time:  6.5      Growth:  8.979281630110643   Population:  188.5649142323235
Time:  7.0      Growth:  9.428245711616176   Population:  197.99315994393967
Time:  7.5      Growth:  9.899657997196984   Population:  207.89281794113666
Time:  8.0      Growth:  10.394640897056833  Population:  218.2874588381935
Time:  8.5      Growth:  10.914372941909676  Population:  229.20183178010316
Time:  9.0      Growth:  11.46009158900516   Population:  240.6619233691083
Time:  9.5      Growth:  12.033096168455415  Population:  252.69501953756372
Time:  10.0     Growth:  12.634750976878188  Population:  265.3297705144419

PROCESSING COMPLETE.
```

Assessments

- No assessments this week
- The next assessment will be held during the practicals in Week 3 – Practical 3
- It will be a short practical test using the lab computers

- Everyone should be able to get 100%!

Practical Test 1

- Held during your practical in Week 3
- You will continue with Practical 3 after the test
- Worth 3% each
- I expect everyone to be able to get 100%

Tasks

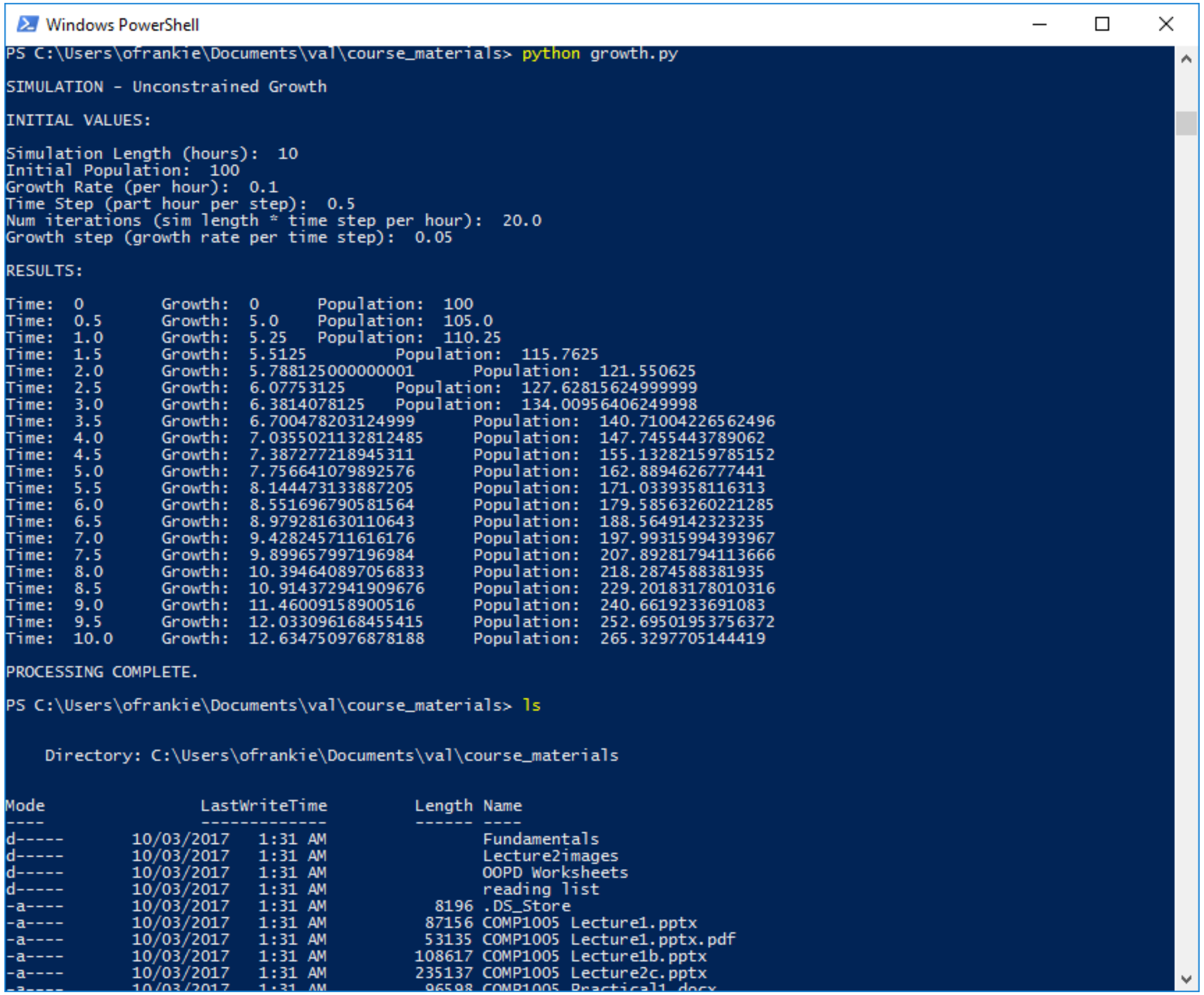
1. Create files and directories as instructed
2. Create Python program to match the description given
3. Capture your command history into a file within the PracTest1 directory
4. Zip your files and submit them through this page

WORKING AT HOME

Anaconda and Windows

(Macs are easy!)

Windows PowerShell



ls command works!

cd

dir

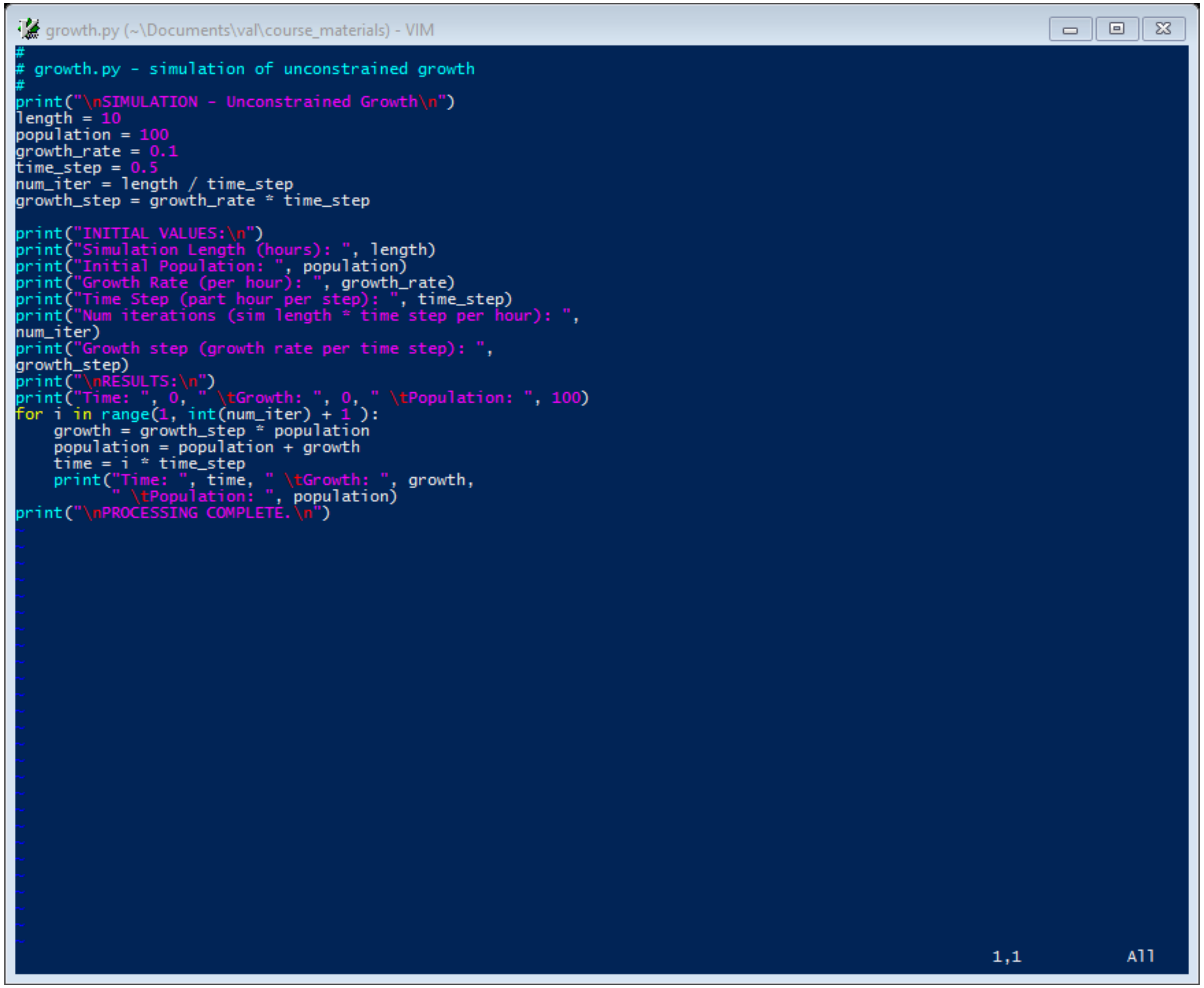
mkdir or md

/ or \ in paths

Root directory is:
c:\Users\username\

Use “python”,
not “python3”

Install gvim/vim 8.0



www.vim.org

gvim80.exe

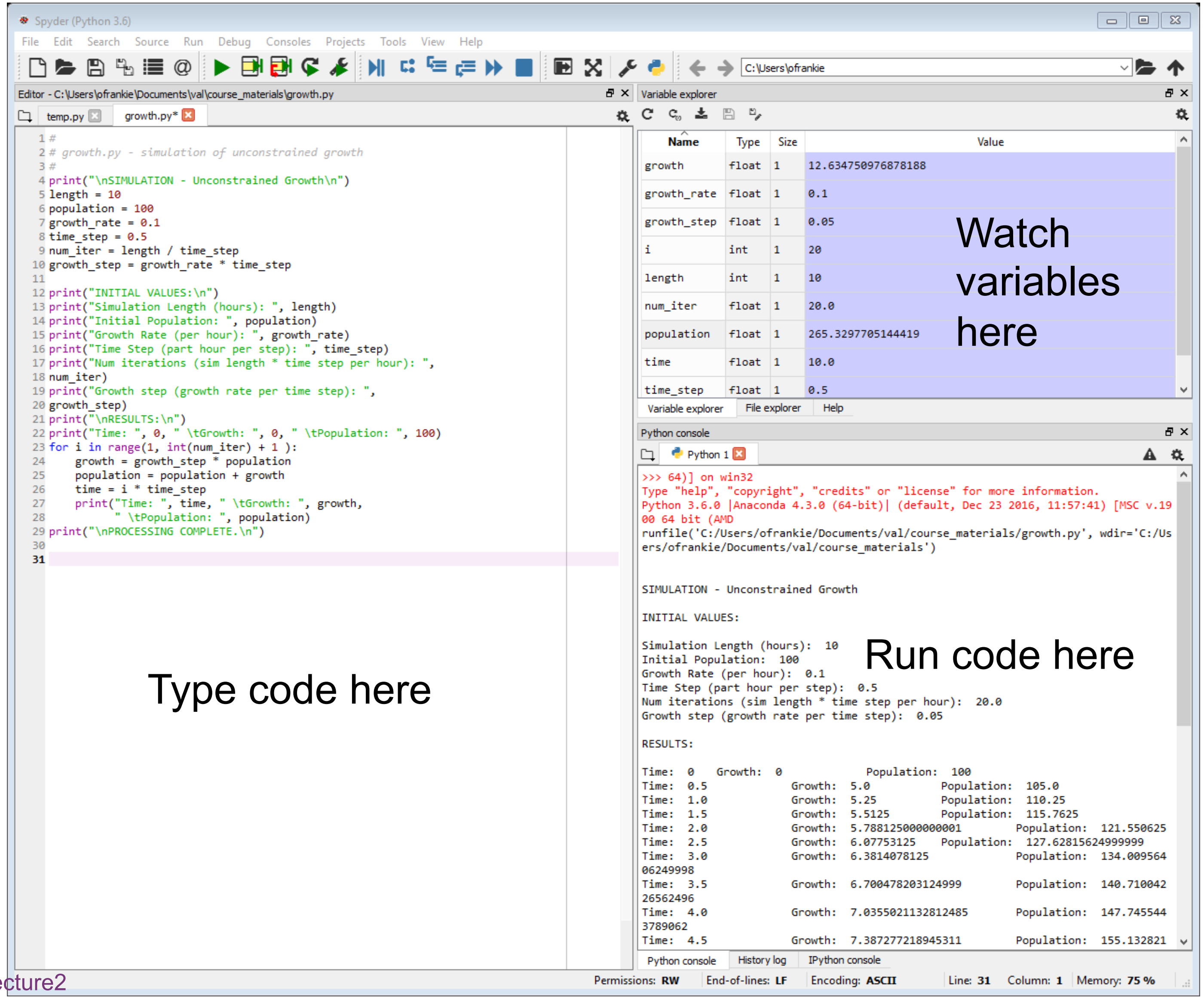
Run from command line :

gvim growth.py

or by right-clicking and selecting the gvim app

Make sure you tick the bat files checkbox

Have a play with Spyder…



On a Mac

- Open “Terminal” application
- Find it with a spotlight search
- The Terminal command line is a variant of Unix – so all the commands will work

References

- Quotes from Monty Python's:
  - Flying Circus
    - http://www.montypython.net/scripts/spam.php
  - The Holy Grail
  - The Life of Brian
- MPI and OpenMP training, Pawsey Supercomputing Centre – random numbers and Method of Darts (by Rebecca Hartman-Baker)
- https://pymotw.com/3/random/index.html
- https://docs.python.org/3/library/stdtypes.html

Next week…

- Arrays and plotting
  - Numpy
  - Matplotlib