Teaching London Computing

A Level Computer Science

Topic 3: Advanced Programming in Python

Queen Mary University of London

King's College London

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE

Department for Education

SUPPORTED BY
MAYOR OF LONDON

NETWORK OF EXCELLENCE COMPUTER SCIENCE TEACHING
UNIVERSITY PARTNER

William Marsh
School of Electronic Engineering and Computer Science
Queen Mary University of London

Aims

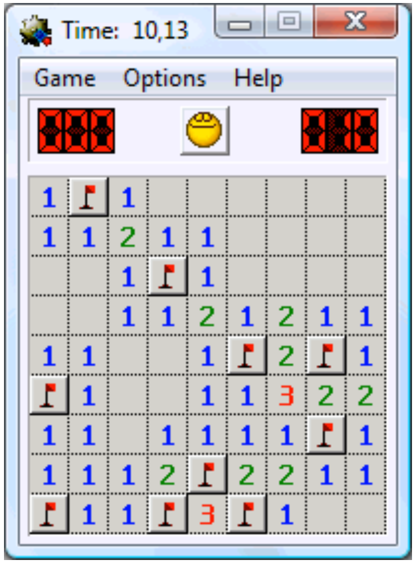
- Further topics in programming
  - Some Python-specific

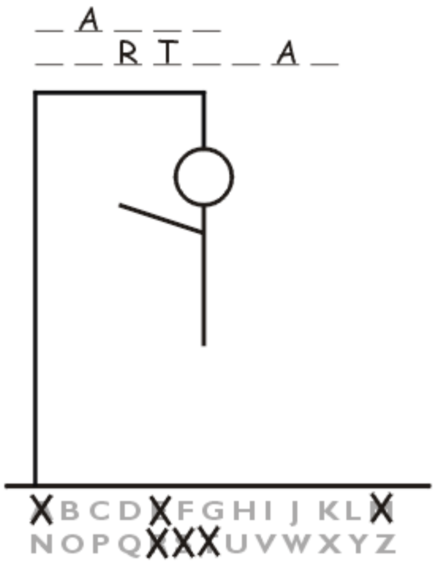
- Representing information
  - Arrays of multiple dimensions
  - Python built-in types: tuples, dictionaries; sequences

- Exceptions: dealing with errors

Two Representation Problems

- Minesweeper squares
  - Flagged / tested / hidden
  - Mine / no mine
  - Number of neighbouring mines





- Hangman words
  - Letter at each position
  - Letters used
  - Letters uncovered

Hangman Example

- Representation changes program

Letter	Positions
A	5
L	6
N	1
S	3
U	0, 2, 4

Complete word

```
[ 'U', 'N', 'U', 'S', 'U', 'A', 'L']
```

Letters guessed

```
[ 'A', 'E', 'T', 'S', 'R']
```

Current display

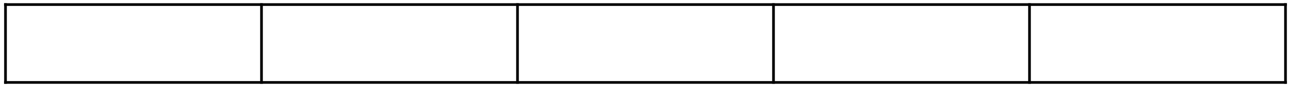
```
[ '_', '_', '_', 'S', '_', 'A', '_']
```

Arrays of Multiple Dimensions

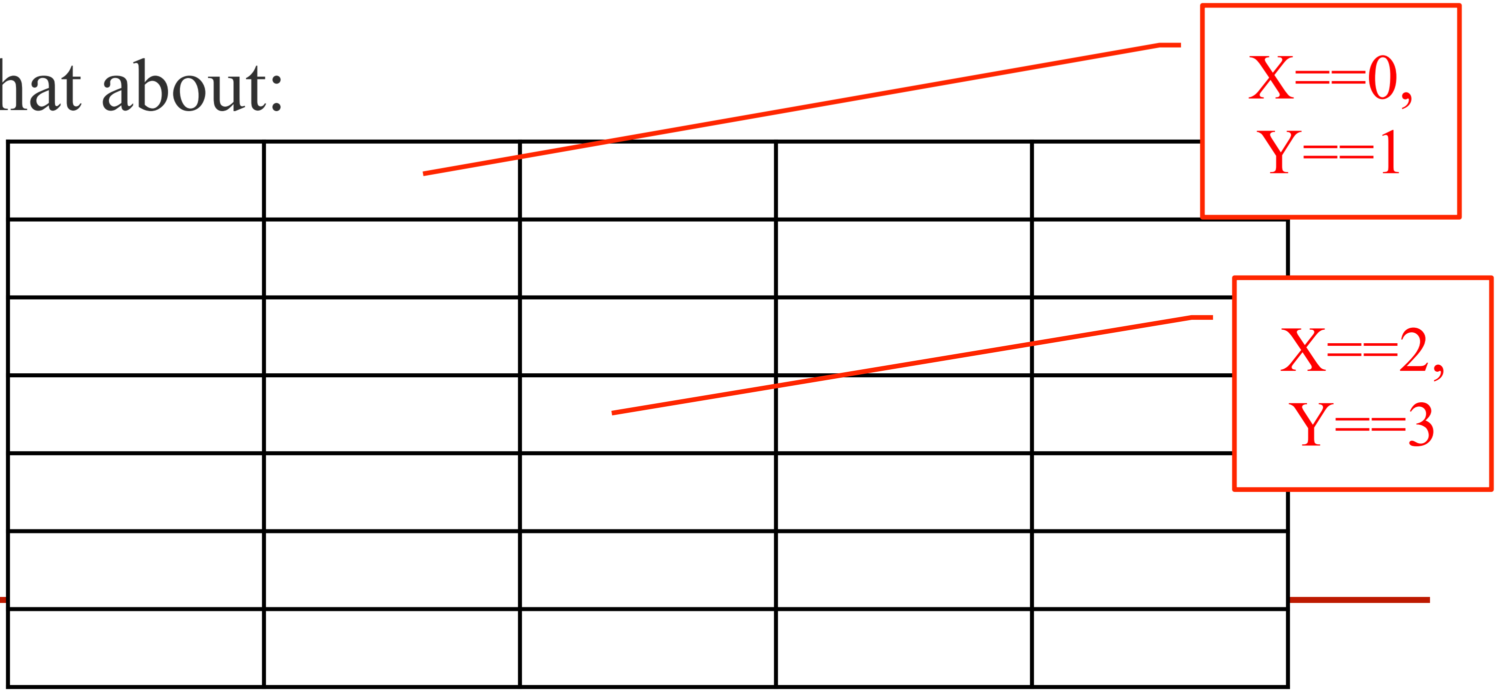
Standard part of A Level

Multidimensional Arrays

- *Recall that Arrays are Lists in Python*
- So far: arrays represent



- What about:



Why Arrays?

- An array is a sequence of memory locations
  - Simple and fundamental idea
  - Really, lists are represented using arrays

Arrays	Lists
Fixed number of entries	Can be extended
All entries the same size	Can have different entries
Continuous in memory	… more complex
Regular shape	Can be irregular

- We use lists to learn about arrays

Table of Data

- Sum the columns in a table of data

10	27	23	32
31	44	12	65
15	17	18	23
??	??	??	??

- Issues
  - Representation
  - Algorithm

Table Representation

- Table represented by list of lists

```
table = [ \
    [10, 27, 23, 32], \
    [31, 44, 12, 65], \
    [15, 17, 18, 23]  \
    [ 0,  0,  0,  0]  \
        ]
```

- Quiz
  - `table[0][1] == ?`
  - `table[1][2] == ?`

Printing a Column

- Two methods

```
def printCol1(table, colN):
    string = ""
    for rowN in range(0, len(table)):
        string += str(table[rowN][colN]) + " "
    print(string)
```

Use two indices

```
def printCol2(table, colN):
    string = ""
    for row in table:
        string += str(row[colN]) + " "
    print(string)
```

Select List from Table

Exercise: Sum Columns

- Adapt the code on the previous slide to print the sum of:
  - A given column
  - Of all columns

Built in Types in Python

- Important in Python programming
- Very useful
- Details specific to Python; related concepts elsewhere

Overview

- Lists – [1,2,3,4]
  - Ordered collection of items; often of the same type.
  - Can be changed (*mutable*)
- Tuples – (1,2,3,4)
  - Immutable ordered collection; often different types
- Ranges – range(1,5)
  - Number sequence; used in for loops
- Sets – {1,2,3,4}
  - Unordered non-repeating collection
- Dictionaries – {1:'one', 2:'two', 3:'three'}
  - Mappings

Tuples – Examples

- Convenient for returning multiple values from a function

```
def getTwo():
      ms = input("A string> ")
      nm = input("A number> ")
      return((ms, int(nm)))

>>> getTwo()
A string> Hello
A number> 99
('Hello', 99)
```

- Unpack

```
>>> t = ("a", 1, [1])
>>> x,y,z = t
>>> z
[1]
```

Assign to multiple variables

Ranges – Examples

- In a for loop:

```
for x in range(1,10,2):
      print("x =", x)

x = 1
x = 3
x = 5
x = 7
x = 9
```

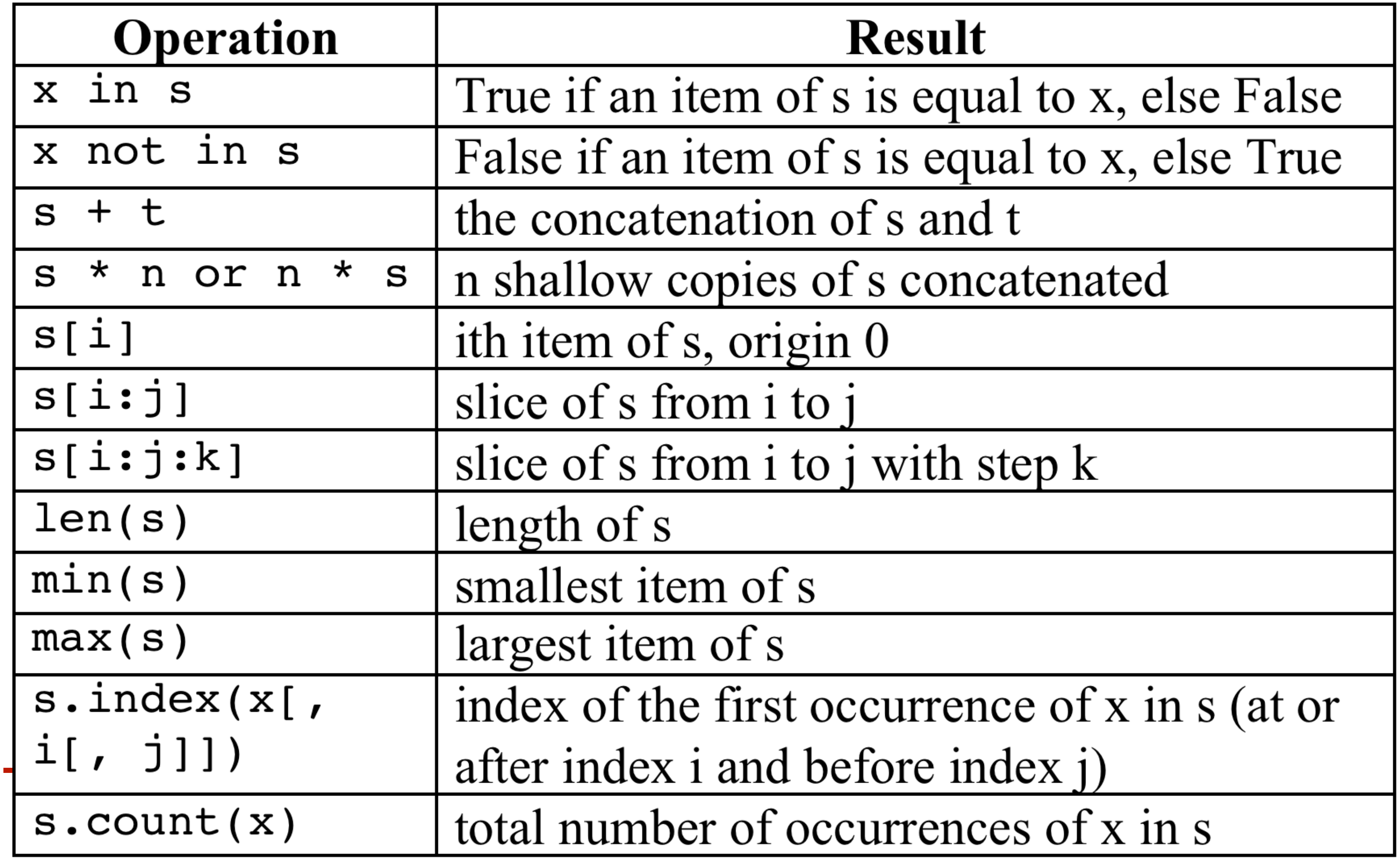
- Convert to a list

```
>> list(range(0,-10,-1))

[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Sequences

- Strings, lists, tuples and ranges are all **sequences**



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`	n shallow copies of s concatenated
`s[i]`	ith item of s, origin 0
`s[i:j]`	slice of s from i to j
`s[i:j:k]`	slice of s from i to j with step k
`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

Mutable and Immutable

- Mutable == can change
  - e.g. append an item to a list
- Immutable == cannot change
  - Concatenating two lists does not change them
  - Copied when necessary

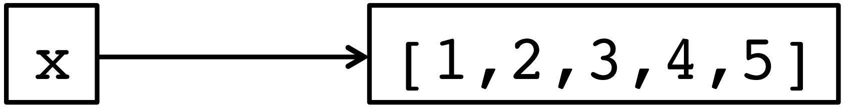
- Lists, sets and dictionaries are mutable
- Strings, tuples and ranges are immutable

Sequences – Mutable Only

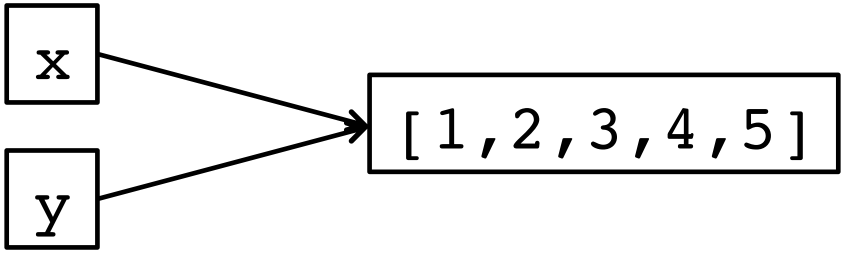
Operation	Result
`s[i] = x`	item i of s is replaced by x
`s[i:j] = t`	slice of s from i to j is replaced by the contents of the iterable t
`del s[i:j]`	same as s[i:j] = []
`s[i:j:k] = t`	the elements of s[i:j:k] are replaced by those of t
`del s[i:j:k]`	removes the elements of s[i:j:k] from the list
`s.append(x)`	appends x to the end of the sequence (same as s[len(s):len(s)] = [x])
`s.clear()`	removes all items from s (same as dels[:])
`s.copy()`	creates a shallow copy of s (same as s[:])
`s.extend(t)`	extends s with the contents of t (same as s[len(s):len(s)] = t)
`s.insert(i, x)`	inserts x into s at the index given by i(same as s[i:i] = [x])
`s.pop([i])`	retrieves the item at i and also removes it from s
`s.remove(x)`	remove the first item from s where s[i]== x
`s.reverse()`	reverses the items of s in place

Understanding Assignment

- Variables (and parameters) **refer** (or **point**) to objects
- Assignment (and function parameters) **copy references**



```
y = x # assignment
```



Other Languages

- Issue: copying large objects (long arrays)

> In Visual Basic, you can pass an argument to a procedure by value or by reference. This is known as the passing mechanism, and it determines whether the procedure can modify the programming element underlying the argument in the calling code. The procedure declaration determines the passing mechanism for each parameter by specifying the ByVal or ByRef keyword.
>
> Quoted from http://msdn.microsoft.com/en-gb/library/ddck1z30.aspx

If an object is immutable, you cannot tell whether it is copied or referenced

Sets and Dictionaries

- Set: a collection of unique objects
  - Not ordered
  - Mutable (but elements must be *immutable*)

- Dictionary: a map from a key to a value
  - Unique key
  - Mutable (key must be *immutable*)

Set Examples

```
>>> s = {1,2,3}
>>> t = set(range(2,11,2))
>>> t
{8, 2, 10, 4, 6}
>>> u = s.union([1,1,1])
>>> u
{1, 2, 3}
>>> u = s.intersection(t)
>>> u
{2}
>>> len(s)
3
>>> {2,4}.issubset(t)
True
>>> s.issubset(t)
False
>>>
```

Making sets

Set operations

Dictionary Examples

```
>>> d1 = {'milk':2,'eggs':6,'tea':1}
>>> d1
{'eggs': 6, 'tea': 1, 'milk': 2}
>>> len(d1)
3
>>> 'books' in d1.keys()
False
>>> 'records' in d1.keys()
False
>>> d2 = dict([((0,0),'B'),((0,1),'G'),((1,0),'B'),
((1,1),'G')])
>>> d2
{(0, 1): 'G', (1, 0): 'B', (0, 0): 'B', (1, 1): 'G'}
>>> d2[(1,1)]
'G'
>>> d1['milk']
2
```

Making a dictionary

Check keys

Tuple as a key

Dictionary look up

Dictionaries versus Arrays

Standard Array	Python Dictionary
Index by number	Key can be a string, pair, …
Indices continuous e.g 0 → 10	Gaps ok
Fixed length	Can add and delete entries
Simple values: number, character	Any value – even a dictionary

- In other languages, library has ‘dictionary’ data structure

Exercise

- Suggest two representations each for minesweeper and / or hangman
  - Write Python to create examples
  - Write Python to update the state
    - New location in the mine field tested
    - New letter guessed in hangman

Exceptions

What Happens When a Problem Occurs

Exception – Example

- `int("XYZ")` – leads to an error
  - Not a programming error: user input
  - Program stops:

```
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    int("xyz")
ValueError: invalid literal for int() with base 10:
'xyz'
```

Error or “exception” name

Exceptions – Trying it out

- Try out the code
- Certain errors possible
- “Catch” the error (i.e. exception) if it occurs and
- … run code to “handle” the error.

- Words
  - “Exception” – a type of error, with a name
  - “Handle” – respond to the error nicely
  - “Catch” – jump to error-handling statement

Exception – Syntax

- Example

```
try:
    in_str = input("Enter a number> ")
    in_num = int(in_str)
except ValueError:
    print("Sorry", in_str, "is not an integer")
```

Two new keywords

Statements where exceptions may occur

Only if exception occurs

When to Use Exceptions

- Robust code: check for errors
  - Why?

- **Either**: Check error cannot occur
- **Or**: Catch exceptions

- Exceptions used:
  - User input
  - OS operation (opening a file)
  - When using library code

Summary

- Representing data
  - Aspect of problem solving
  - Easier in Python: build in ‘data structures’

- Handle exceptions for robust code