A Level Computer Science

Topic 3: Advanced Programming in Python
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

<table>
<thead>
<tr>
<th>Letter</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>L</td>
<td>6</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>3</td>
</tr>
<tr>
<td>U</td>
<td>0, 2, 4</td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

- What about:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

X==0, Y==1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

X==2, Y==3
Why Arrays?

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

<table>
<thead>
<tr>
<th>Arrays</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed number of entries</td>
<td>Can be extended</td>
</tr>
<tr>
<td>All entries the same size</td>
<td>Can have different entries</td>
</tr>
<tr>
<td>Continuous in memory</td>
<td>… more complex</td>
</tr>
<tr>
<td>Regular shape</td>
<td>Can be irregular</td>
</tr>
</tbody>
</table>

- We use lists to learn about arrays
### Table of Data

- Sum the columns in a table of data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>27</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>31</td>
<td>44</td>
<td>12</td>
<td>65</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>??</td>
<td>??</td>
<td>??</td>
<td>??</td>
</tr>
</tbody>
</table>

### Issues

- Representation
- Algorithm
Table Representation

- Table represented by list of lists

```python
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

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

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

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

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

• Unpack

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

Assign to multiple variables
Ranges – Examples

• In a for loop:

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

```none
x = 1
x = 3
x = 5
x = 7
x = 9
```

• Convert to a list

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

```none
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```
Sequences

- Strings, lists, tuples and ranges are all sequences

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x in s</code></td>
<td>True if an item of s is equal to x, else False</td>
</tr>
<tr>
<td><code>x not in s</code></td>
<td>False if an item of s is equal to x, else True</td>
</tr>
<tr>
<td><code>s + t</code></td>
<td>the concatenation of s and t</td>
</tr>
<tr>
<td><code>s * n</code> or <code>n * s</code></td>
<td>n shallow copies of s concatenated</td>
</tr>
<tr>
<td><code>s[i]</code></td>
<td>ith item of s, origin 0</td>
</tr>
<tr>
<td><code>s[i:j]</code></td>
<td>slice of s from i to j</td>
</tr>
<tr>
<td><code>s[i:j:k]</code></td>
<td>slice of s from i to j with step k</td>
</tr>
<tr>
<td><code>len(s)</code></td>
<td>length of s</td>
</tr>
<tr>
<td><code>min(s)</code></td>
<td>smallest item of s</td>
</tr>
<tr>
<td><code>max(s)</code></td>
<td>largest item of s</td>
</tr>
<tr>
<td><code>s.index(x[, i[, j]])</code></td>
<td>index of the first occurrence of x in s (at or after index i and before index j)</td>
</tr>
<tr>
<td><code>s.count(x)</code></td>
<td>total number of occurrences of x in s</td>
</tr>
</tbody>
</table>
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

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

• Variables (and parameters) refer (or point) to objects

• Assignment (and function parameters) copy references
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.


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

```python
>>> 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
>>> 
```
>>> 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
## Dictionaries versus Arrays

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

- 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

```python
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