IFB104 — Building IT Systems
Topic 4 — How to Think Like a Computer

School of Computer Science
Semester 1, 2025

Assignment 1B – Where to Start

- You can address the tasks in any order you like.
- You can also use another file to work on developing your functions, and move them to your assignment with the right template once they are working.

1. *make_book_list* function+ unit tests

2. *title_length* function+ unit tests

3. *interact* function. Just print ("This function is currently under development") instead of calling up the functions you have not yet written, or use your default values from1A.

You can now "pass" the first three criteria, and full marks on the last!

4. *make_index*, *word_occurrences*, *search_word,* and *interact* function again.

Need help?

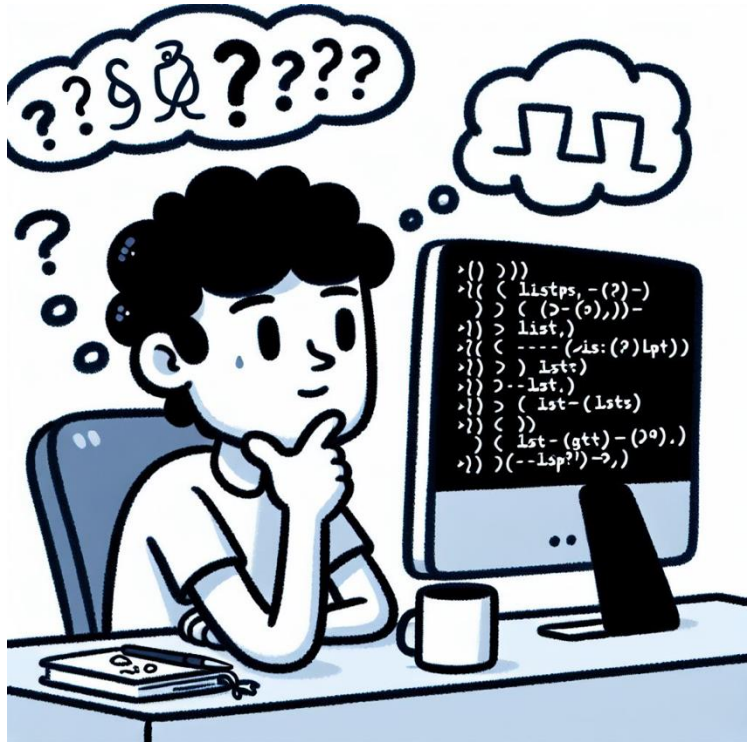
- There is one *Student Success Group* special workshops for IFB104 students before the deadline
  - Wednesday afternoon on-campus
- Go to the SSG site for full details and to register: https://unihub.qut.edu.au/students/events/search?Text=ifb104
- See Canvas *Unit Overview* | *Getting Help* for links to all the SSG IFB104 help services

Need help?

- *STIMulate* Peer Learning Facilitator consultation sessions
  - On-campus, Monday to Friday, 10am to 3pm – see the drop-in timetable at https://stimulate.qut.edu.au/ (choose “IFB104” as the unit)
  - Online – submit a request for peer support via https://qut.to/q826v

Aims of this week's lecture

- Consolidate your understanding so far.
  - Working through an example of lists of lists.
  - Looking at structures a bit different to lists.
  - Try different ways to think before writing the code.
  - Working through a few typical exam questions.



Example: Creating a List of Lists

- Input: on a single line, a list of animals is provided. For each animal, its name is provided, followed by its height (in cm) and weight (in kg). The list is provided on a single line, with values separated by spaces. For example:

```
unicorn 120 190 mouse 8 0.2 dragon 300 100
```

- Output: A list of lists, with each list having the name of the animal, and a list of measures. For example: [['unicorn', ['120', '190']],…]
- Write an interactive program that asks a use to enter the list and then provides the formatted version, using a function for the formatting.
- Write unit tests for the function.

```
len(L)
```
returns the length of list *L*

```
from doctest import testmod
testmod(...)
```

```
from re import findall
findall('m[aeiou]{3} ', var_name)
```

```
L.append(i)
```
adds item *i* onto the end of list *L*

Demonstration files: `list_list_interactive.py, list_list.py`

Part A – Tuples and Dictionaries

Tuples

- Tuples are similar to list in that they contain sequences of values.
  - Example: animals=(turtle, cat, camel)
- However, they are **Immutable**: Tuples cannot be modified after creation. Once created, their elements cannot be changed.
- Ideal for collections of items that should not change, ensuring data integrity.

Feature	List	Tuple
Mutability	Mutable	Immutable
Syntax	[]	()
Example	[1, 2, 3]	(1, 2, 3)
Use Case	Dynamic collections	Fixed collections

Some Function return tuples

- Findall() returns tuples if several parts of the regex are captured
  - Example:

```
>>> input_text = "unicorn 120 190 mouse 8 0.2 dragon 300 1000"
>>> elements = findall('([a-z]+) ([0-9\.]+) ([0-9\.]+)',input_text)
   [('unicorn', '120', '190'), ('mouse', '8', '0.2'), ('dragon', '300',
   '1000')]
```

- To convert a tuple into a list: list(*your_tuple*)

```
>>> list(elements[0])
      ['unicorn', '120', '190']
```

Dictionaries

- Also store collections of items, but
  - **Unordered**: Elements in a dictionary do not have a defined order. There is no index to retrieve items
  - **Key-Value Pairs**: Access elements by their keys, not by index.
  - They are also **mutable**: Can be modified after creation (e.g., adding, removing, or changing key-value pairs).

Feature	List	Dictionary
Structure	Elements	Key-Value Pairs
Access	By Index	By Key
Order	Ordered	Unordered (insertion order in Python 3.7+)
Syntax	[]	{}
Example	[1, 2, 3]	{'key': 'value'}

Dictionaries

- They are defined using curly braces {}. Elements are retrieved using the key.

```
>>> animal_dict = {'unicorn':[120, 190], 'mouse':[8, 0.2],
'dragon':[300, 1000]}

>>> animal_dict['unicorn']

    [120, 190]
```

```
>>> for key in animal_dict:
        print(key)
    unicorn
    mouse
    dragon
```

```
>>> for value in animal_dict.values():
        print(value)
    [120, 190]
    [8, 0.2]
    [300, 1000]
```

Part B – Thinking Tools

Algorithms

- Set of instructions to automate a process.

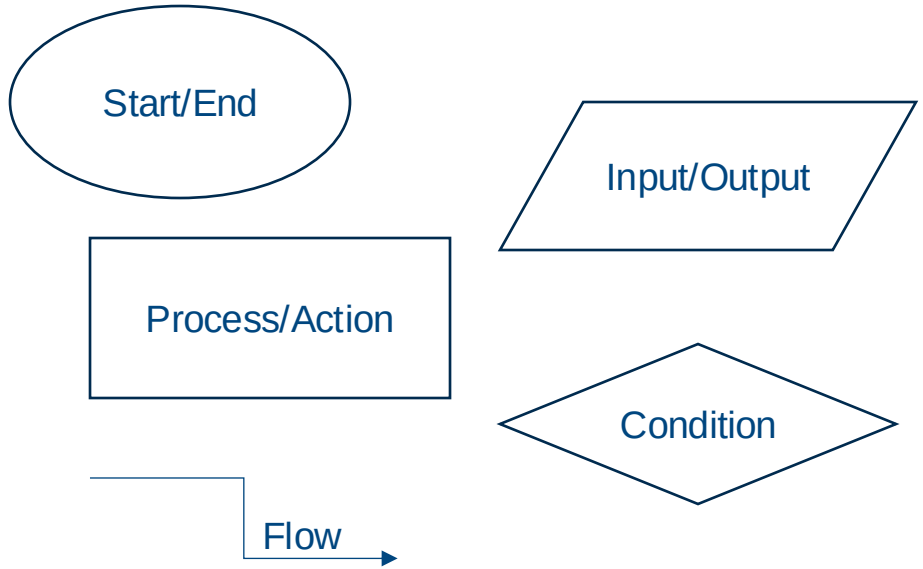
- To automate, you need to understand the process first!

Try it by hand first

- See video recording of the week 6 lecture!

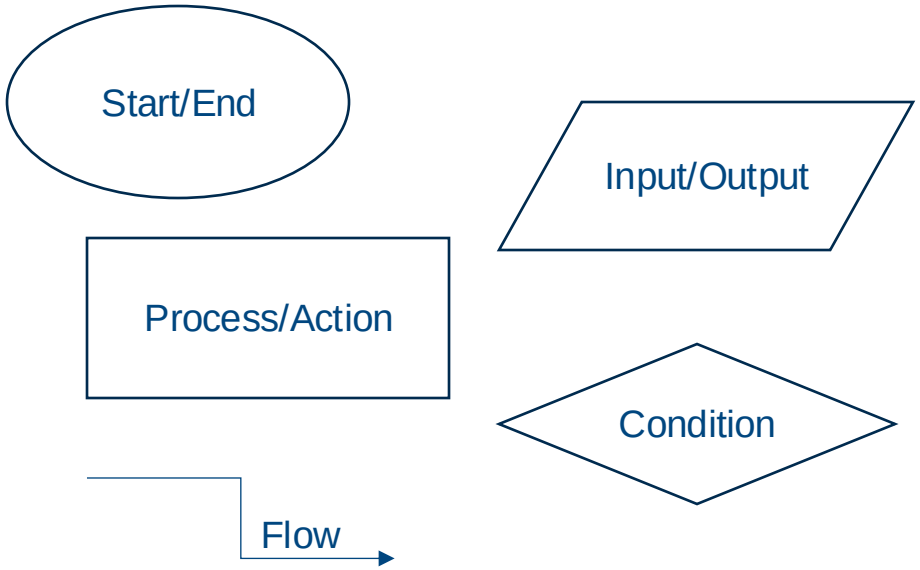
Flow Charts

- A flowchart is a visual representation of a process or algorithm using symbols and arrows.
- It helps in understanding, analyzing, and communicating the steps in a process.
- **Basic Symbols**:
  - **Oval**: Start/End
  - **Parallelogram:** Input/Output
  - **Rectangle**: Process/Action
  - **Diamond**: Decision/Condition
  - **Arrow**: Flow of control



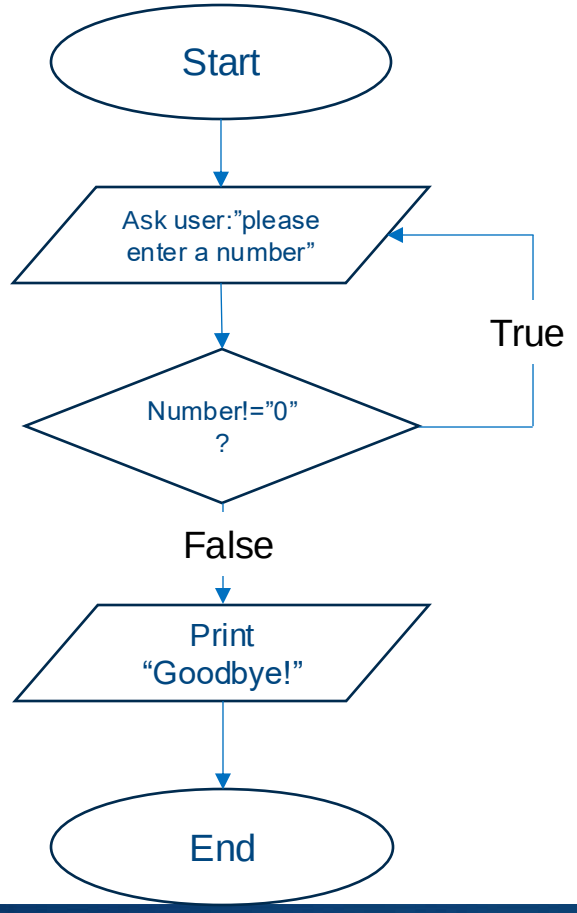
Flowchart: Interactive Loop

- Write a program that asks the user to enter numbers repeatedly until the user enters “0”



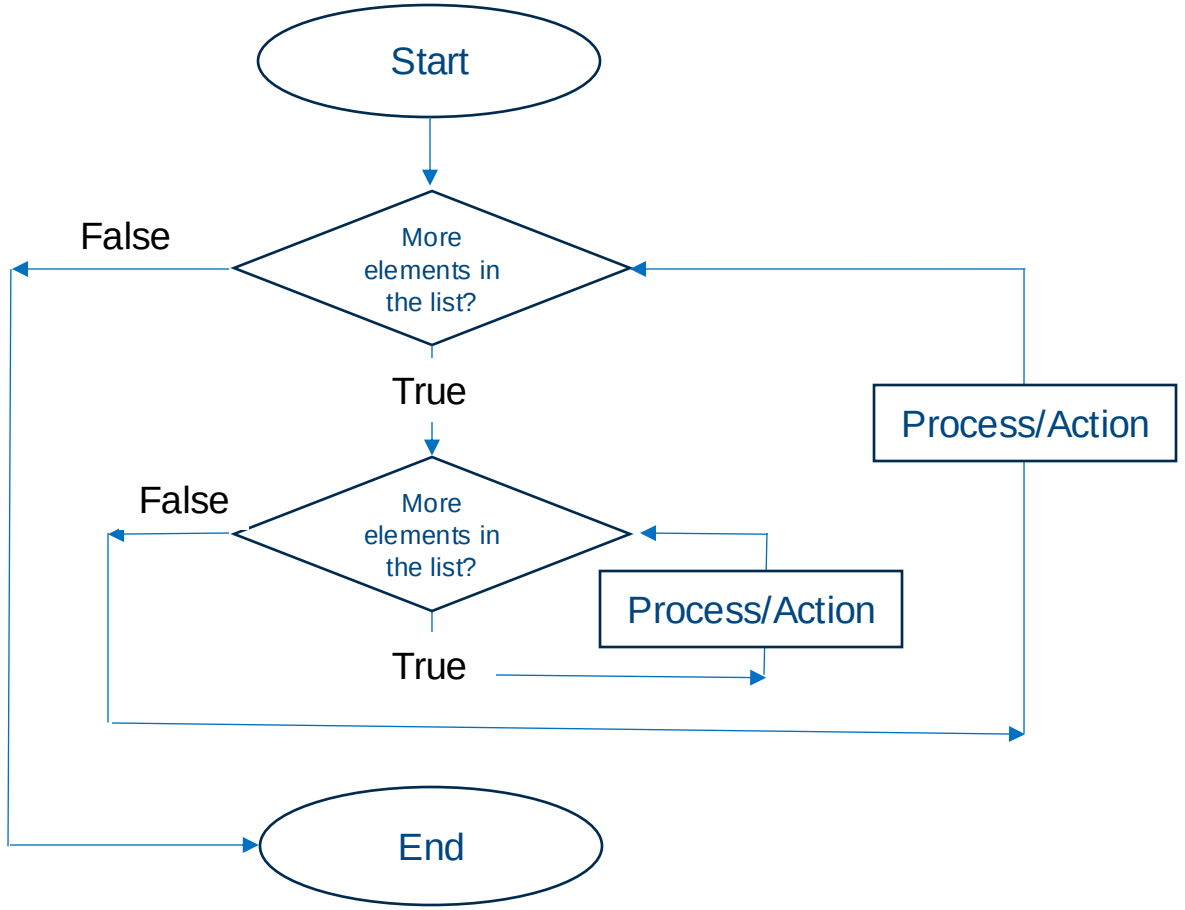
Flowchart: Interactive Loop

- Write a program that asks the user to enter numbers repeatedly until the user enters “0”



FlowChart: Nested loops for lists of lists

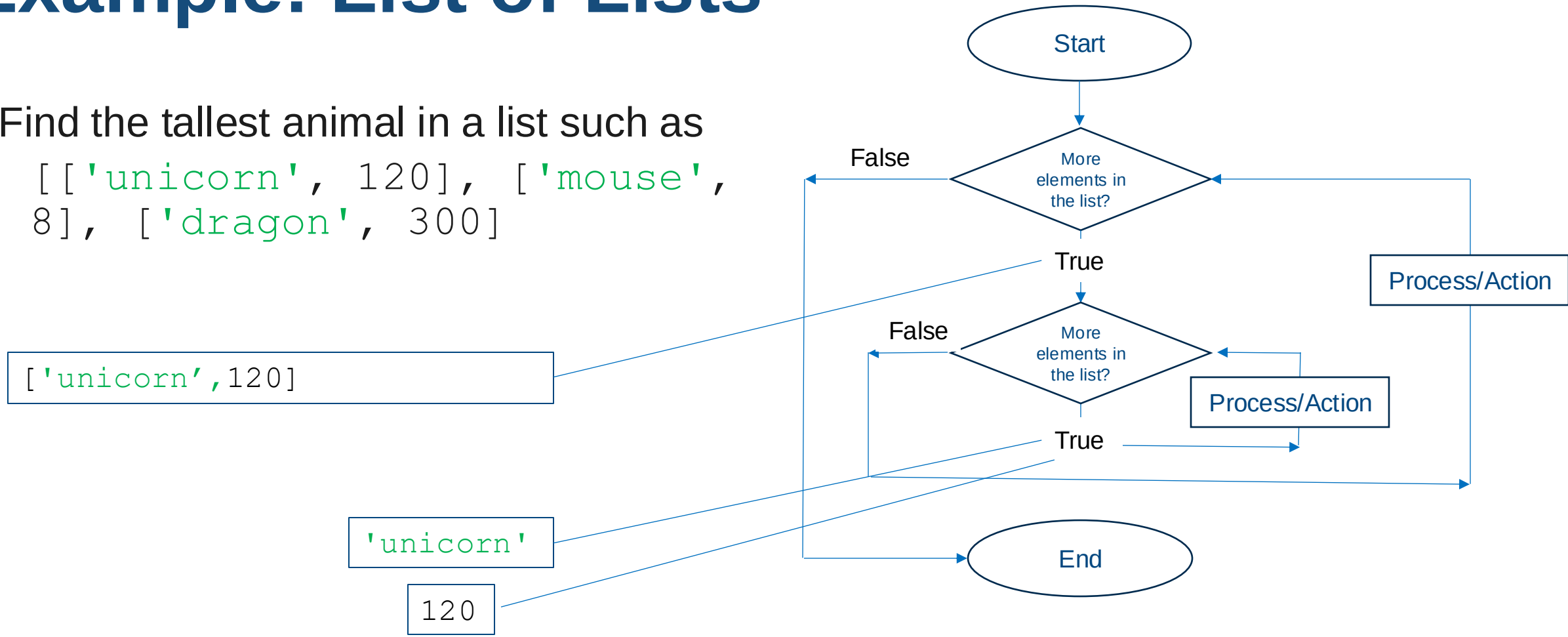
- Step by Step



Example: List of Lists

- Find the tallest animal in a list such as

```
[['unicorn', 120], ['mouse',
8], ['dragon', 300]
```



Useful Modules and Functions

The random module

- So far we have been looking at ‘core’ features of the Python language
- To extend the language’s capabilities we can import additional *modules* containing extra functions
  - Before using an external module we must *import* the functions we want to use
- A module that helps us create simple game-like programs is the `random` module for creating random numbers and choosing random values from lists
  - See the *Python Standard Library* manual under *Generate Pseudo-Random Numbers* for more detail

```
>>> # import two functions from the random module
>>> from random import randint, choice
>>> randint(3, 9) # choose a no. between 3 and 9 (incl.)
3
>>> randint(3, 9) # choose another
5
>>> randint(3, 9) # and another
9
>>> options = ['a', 'b', 'c'] # create a list
>>> choice(options) # choose a random value from the list
'c'
>>> choice(options) # choose another
'b'
>>> choice(options) # and another
'b'
>>>
```


Demonstration file: `random_numbers.py`

Totally random?

Part C – Exam Review

Lists, evaluatio

Sequences of assignments allow us to break complex calculations up into a series of intermediate steps. Consider the following short sequence of assignments with deliberately obscure variable names. Assume that variables `p` and `q` are both lists each containing two integer values in the range `-100` to `100`, inclusive.

```
first_diff = p[0] - q[0]
second_diff = p[1] - q[1]
abs_one = abs(first_diff)
abs_two = abs(second_diff)
coverage = abs_one + abs_two
```

With respect to the final value of variable `coverage`, which of the following single assignment statements is equivalent to the sequence above, i.e., a statement that performs the same calculation but without using any intermediate variables?

- ○ `coverage = abs(p[0] - p[1]) + abs(q[0] - q[1])`
- ○ `coverage = abs(p[0] - q[0] + p[1] - q[1])`
- ○ `coverage = abs(p[1] - q[0]) + abs(p[0] - q[1])`
- ○ `coverage = abs(p[0] - q[0]) + abs(p[1] - q[1])`
- ○ `coverage = abs(p[0] - q[1]) + abs(p[1] - q[0])`

Conditions

Boolean expressions and conditional statements allow your program code to choose between alternative actions. The exercises and assessment items in this unit have frequently involved such statements.

Consider the following Python statement. Assume that variables `price` and `durability` have already been assigned integer values.

```
if price <= 1000 and durability >= 4:
    outcome = 'Excellent'
elif price > 1000:
    if durability < 4:
        outcome = 'Bad'
    else:
        outcome = 'Fair'
else:
    outcome = 'Okay'
```

Under what conditions from the options below will variable `outcome` be assigned the string 'Fair'?

This is a multiple answer question. You must select all the correct answers below and no others.

- [ ] When `price` equals 1000 and `durability` is greater than 4
- [ ] When `price` is greater than 1000 and `durability` equals 4
- [ ] When `price` equals 1000 and `durability` equals 4
- [ ] When `price` is less than 1000 and `durability` equals 6
- [ ] When `price` equals 850 and `durability` is less than 4
- [ ] When `price` is greater then 1000 and `durability` equals 5
- [ ] When `price` equals 1500 and `durability` is less than 4

Loops

Which of the following nested loops will correctly sum all the elements in a list of lists?

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

A)

```
total = 0
for row in matrix:
    for element in row:
        total += element
print(total)
```

B)

```
total = 0
for i in range(len(matrix)):
    for j in range(len(matrix[i])):
        total += matrix[i][j]
print(total)
```

C)

```
total = 0
for row in matrix:
    total += sum(row)
print(total)
```

D) All of the above

Regular Expressions

- Given the string text = "Contact us at support@example.com or sales@example.org.", write a regular expression that will extract all the email addresses

Before next week ...

- Submit Assessment Task 1B!

- Now you have all the foundations, we are moving to part 2 of the unit, with Madison!
  - Graphical Interfaces
  - Interacting with external data
  - Preventing crashes
  - Understanding broad IT systems

- Continue sending enquiries to ifb104.query@qut.edu.au,
  or through the Canvas Discussion board.