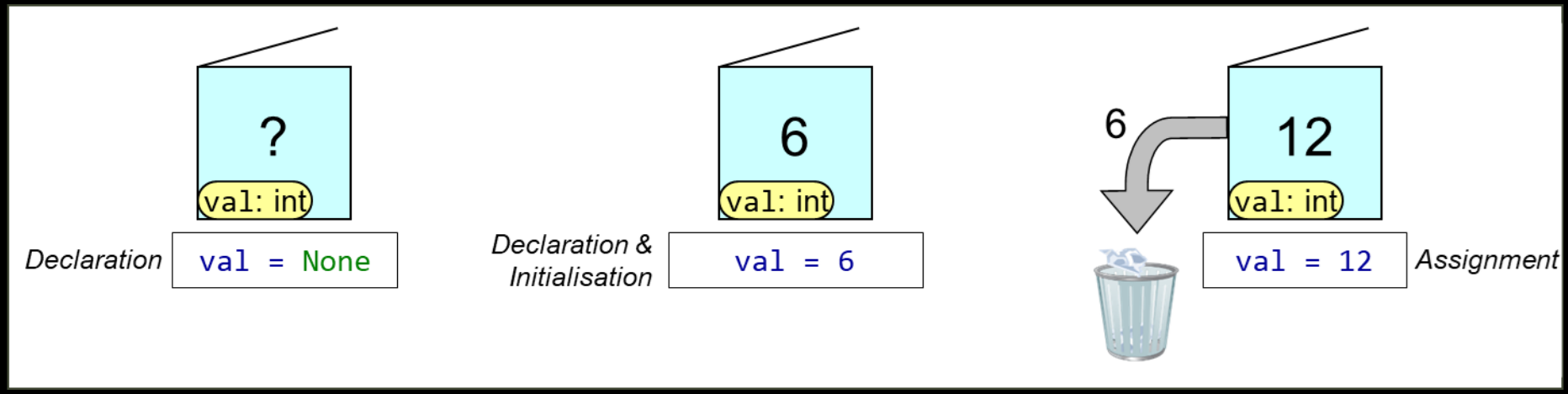
COM4013
Introduction to Software Development

- Week 6
- Lists and Code Quality

- *Umar Arif – u.arif@leedstrinity.ac.uk*

Variables: Concept

A variable is a labelled box containing a single value of a given type:



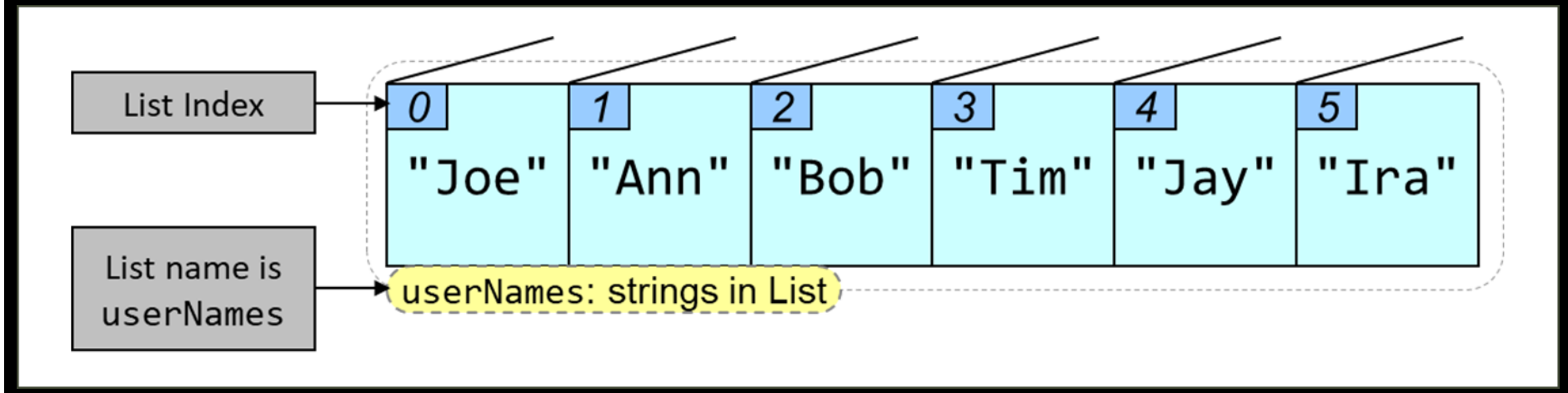
- What if we need ***several*** related values?
  - Perhaps create several variables:
    - width
    - height
  - Or maybe use this shorthand style:
    - width, height

- But what if we needed 100 related values...

List: Concept

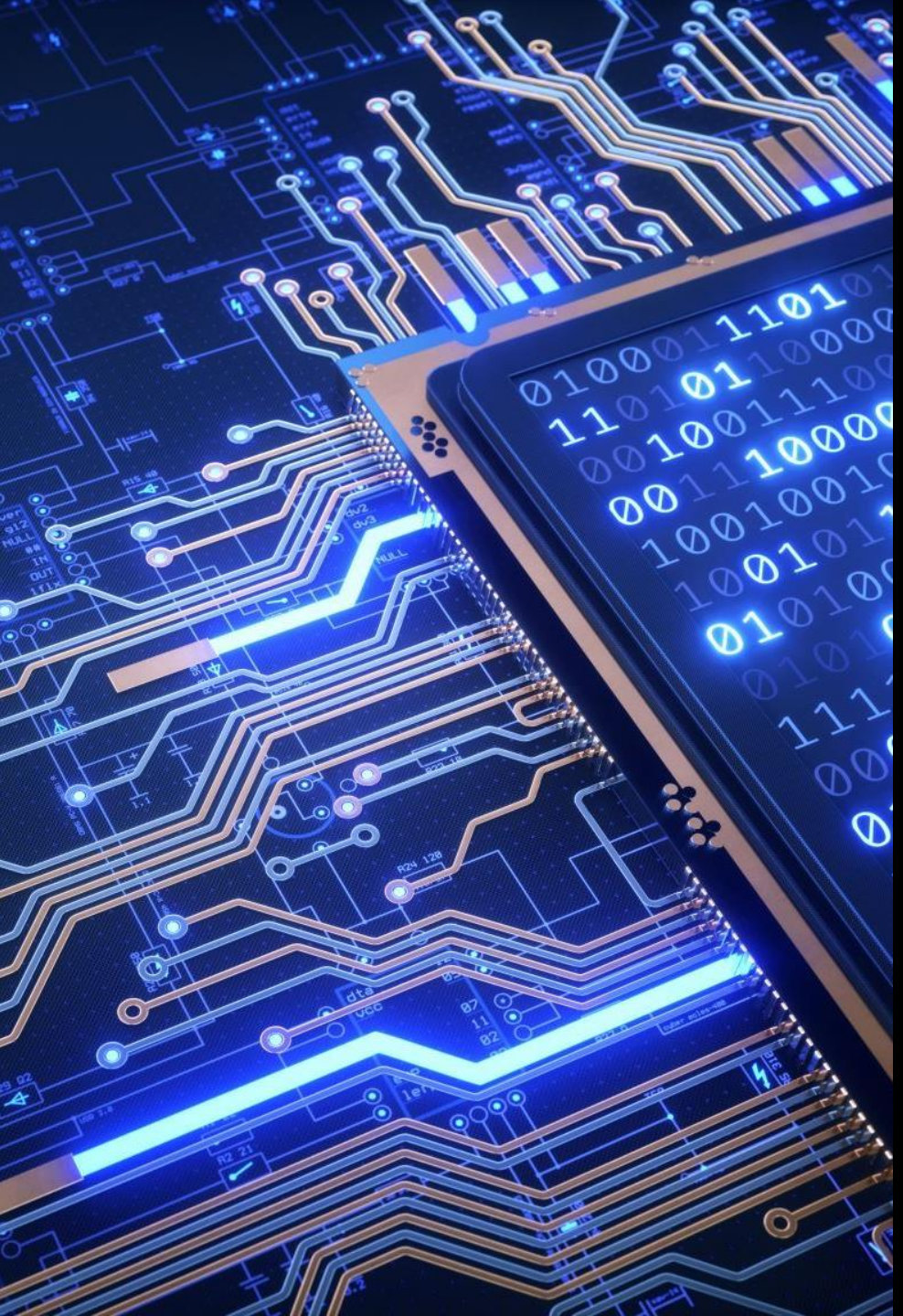
A **List** is a single variable that can store multiple values
Think of it as a set of drawers (or sequence of elements)

- The drawers as a whole have a label, the List name
- Each individual drawer is numbered, starting at 0 (zero)
  - These are called the ***List indexes***



Lists are a type of **collection** or **container**

- There are other types that you may see later in your course



Python Lists Are Different...

- Unlike other languages, Python allows you to create Lists with elements of different types.
  - You can store integers, strings, Booleans, and other data types in Lists.
- In Python, Lists are more flexible regarding their size. Unlike other languages, Python Lists are not limited to a fixed number of "drawers." You can change the size of a List dynamically by adding or removing elements. There is no predefined length for a List; it can grow or shrink as needed.
  - You can use `len(containerVariableName)` to get length of a List.
- Lists, as well as other containers, are used to store collections of items. For example:
  - A List of user names
  - A List of computer network addresses
  - A List of file names

Creating a List

Declare a List in Python using square brackets [ ]

```
myList = [1, 2, 3, 4, 5] # A List with 5 values

listLength = len(myList)
print(listLength) # This will return 5 for the List above
```

When using len() to get the container length of a List, it returns the number of elements in the List, including the one with index 0. So, if you have a List with five elements, the indices are 0, 1, 2, 3, and 4, and `len()` will return 5.

You can also declare an empty List. You may find that you are not sure about what you're actually going to be storing in your List (web-scraping example).

```
myList = []
```

Creating a List

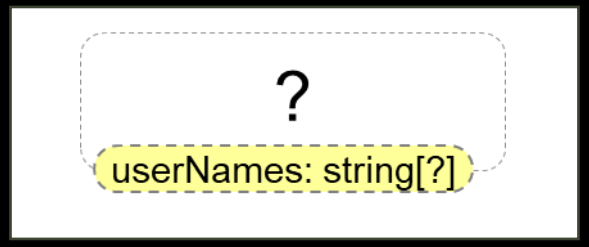
Declare a List and **assign** it with specific values like this:

```
lotteryNumbers = [4, 12, 18, 33, 37, 42]
```

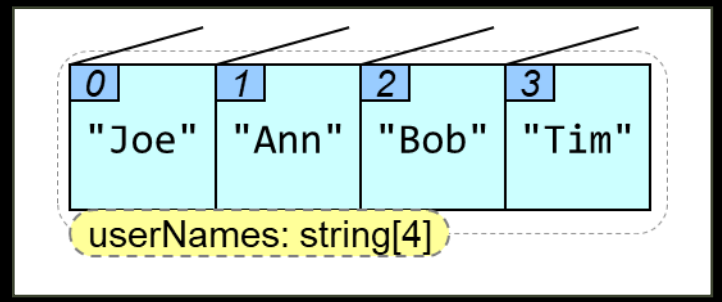
This explicitly sets up each of the 6 integers inside of the `lotteryNumbers` list

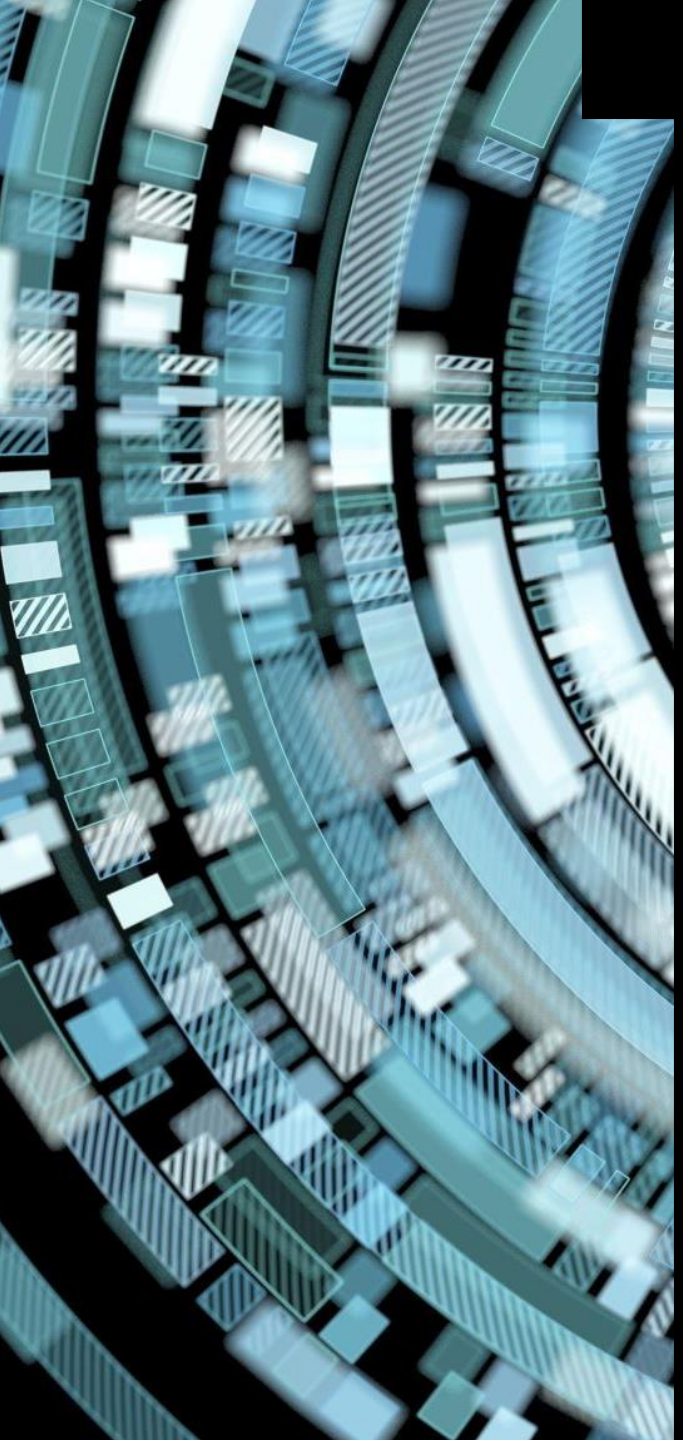
---

```
userNames = []
```




```
string[] userNames = ["Joe", "Ann", "Bob", "Tim"]
```





Accessing List Elements: Indexes

Once we have created a List, we access the individual values using square brackets [ ] and the *List indexes*:

```
lotteryNumbers = [4, 12, 18, 33, 37, 42]
print(lotteryNumbers[0])
print(lotteryNumbers[1])

--Output--
4
12
```

Notice how the first value is accessed with the index 0

- Starting with 1 is a very common cause of bugs for novices

The individual List values are often called the ***List elements***

The code above displays “List element 0” and “List element 1”

Accessing List Elements: Examples

We can use individual List elements like any other variable:

```
userNames = ["Joe", "Jane", "Jack"]
print(f"First user is called {userNames[0]}")

--Output--
First user is called Joe
```

```
grades = [72.0, 56.5, 61.1]

// Add up the grades and divide by 3 to get the average
averageGrade = (grades[0] + grades[1] + grades[2]) / 3 # Why parenthesis?
print(averageGrade)

--Output--
63.2
```

Out of Range Indexes

Since the Lists length is dependent on the number of elements within it, you must make sure you use indexes that are in the correct range for the List:

```
grades = [72.0, 56.5, 61.1]
print(grades[0]) # OK:  72.0
print(grades[1]) # OK:  56.5
print(grades[2]) # OK:  61.1
print(grades[3]) # BUG: IndexError – You could catch this…
```

- This will cause an exception (a ‘crash’)
- This can be confusing because the List was declared with 3 elements, so it seems natural to use index 3 for the last one
  - But we start counting at 0, so the only valid indexes are 0,1,2 here

Be very careful to avoid this kind of error in your own code

- This is an example of an **out-by-one** error

List Methods to Keep in Mind...

To add something to a List we can use the `append()` method. This method adds the new element to the end of the list (thus, it is the last element of the list)

```
students = ["Subhan", "Amna", "Noya"]
students.append("Ben")
# students is now ["Subhan", "Amna", "Noya", "Ben"]
```

If we wish to add something to a specific index, then we can use the `insert()` method

```
students.insert(1, "Jacob")
# students is now ["Subhan", "Jacob", "Amna", "Noya", "Ben"]
```

There are instances where instead of adding a single element to a List we want to append (`extend()`) elements from one List to another

```
NewStudents = ["Nathan", "Josh", "Lewis"]
students.extend(newStudents)
# students is now ["Subhan", "Jacob", "Amna", "Noya", "Ben", "Nathan",
"Josh", "Lewis"]
```

List Methods to Keep in Mind...

To remove something from a List we can use the `remove(value)` method. This method removes the first occurrence of the element from the List

```
students = ["Arefeen", "Manenty", "Josh", "Oliver", "Arefeen"]
students.remove("Arefeen")
# students is now ["Manenty", "Oliver", "Josh", "Arefeen"]
```

If we wish to remove something at the end of the List, we can use the `pop()` method

```
students.pop()
# students is now ["Manenty", "Josh", "Oliver"] - A double takedown
```

We can also remove an element from a specific index. This can be done using the `pop(index)` method

```
students.pop(2) # Bye Ollie
# students is now ["Manenty", "Josh"]
```

Remember that in programming we start counting from 0... And, that there are many other List methods (but I'll let you find those on your own).

‘for’ Loops and Lists

The *fixed* size of Lists (once initialised) makes them ideally suited to a `for` loop:

```
grades = [72.0, 56.5, 61.1]
for i in range(3): # Range function value: start|stop|step?
    print(grades[i])
```

```
--Output--
72.0
56.5
61.1
```

The loop variable index starts at 0 and counts up to 2

- Careful: it doesn’t count to 3 because it says < 3

So first time around the loop index = 0 and this code executes:

```
print(grades[0])
```

Then the next two times around the loop index = 1, then 2

```
print(grades[1])
print(grades[2])
```

‘for’ Loops and Lists

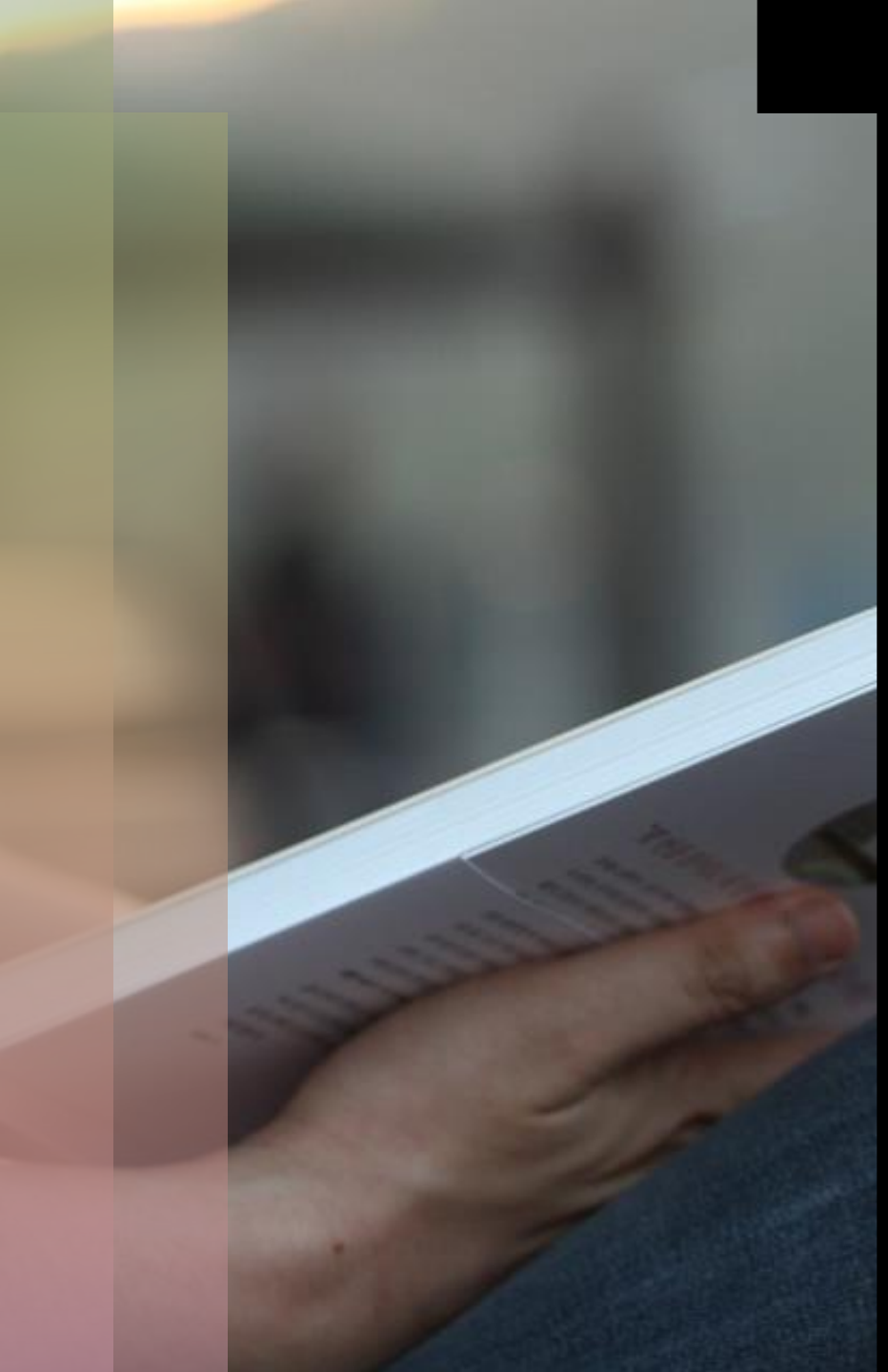
Python Lists **DO NOT** contain a ***property*** specifying their own length or size:

```
userNames = ["Joe", "Jane", "Jack"]
print(userNames.Length) # NOPE – this don’t work
--Output--
3
```

We can instead use the Python `len()` method

```
for i in range(0, len(userNames), 1):
    print(userNames[index]);
--Output--
Joe
Jane
Jack
```

This style of loop is very common when working with Lists in Python



Readability: Indentation

Several times we have focused on ***readability***. There are three readability considerations that are important:

- Variable names, comments and indentation
- We have already looked at variable names and comments

Indentation is using tabs or spaces at the start of each line to push the code in and show the structure:

```
while (userGuess != "tomato"): # Loops while the
user is wrong
userGuess = input("Wrong, guess again: ") # User has
another guess
```

- The lines inside the `while` loop are pushed inwards
- Usually use a single tab, or 4 spaces (either, but be consistent)

Readability: Indentation Examples

Very bad indentation:

- Probably from copy and pasting without tidying up afterwards. It won't work in Python.

```
for i in range(10):
for j in range(10):
print("X", end="")
print("X", end="")
print()
```

Same code, correct indentation (line up as it make it easier to read):

```
for i in range(10):
    for j in range(10):
    print("X", end="")
    print("X", end="")
print()
```

Code Quality: Testing

We always need to test our code. We know what we expect the program to do, and we have looked at debugging already (pdb library and prints). But user input can cause a huge range of unexpected problems. So, we need to test thoroughly.

Question. What kind of mistake can the user make when inputting:

1. Numbers?
2. Text?

- Numbers: Too large and small values, negative values, 0, floating point values, letters or symbols instead of numbers
- Strings: Upper- and lower-case combinations, the empty string (just press return), spaces in strange places
- In both cases, special key combinations (like Alt+F4, or Ctrl+C)!
- Completely robust code needs to consider all these cases (except perhaps special keys)

Code Quality: Test Plans

On other modules you will consider testing in detail (User Centered Design).

At this stage we might want to consider an informal ***test plan***: a list of all possibilities that you go through & check:

Test	Required Outcome	Actual Outcome	Test Result
Main menu: enter option 0 or less	Asks user to re-input repeatedly	Was asked to re-input until correct	PASS
Main menu: enter option 5 or more	Asks user to re-input repeatedly	Was asked to re-input until correct	PASS
Main menu: enter letters instead of number	Asks user to re-input repeatedly	Crash	FAIL
Filtered file listing, enter empty string as filter	Display all files	Displays no files, but continues correctly	FAIL

Summary

In today's lecture we have covered:

- **Lists**
  - A variable that contains a collection of values, all the same type
- **List Indexes**
  - Using integers to access the individual values in a List
- Using loops to access Lists
- **More readability and quality**
  - Indentation
  - Thinking about errors and testing