At the end of this lecture, students should be able to:
Understand that a variable stores a reference to the object
Understand that string objects are a sequence of characters
Use the `len()` function to calculate how many characters are in a string
Obtain a single character from a string
Slice strings
Concatenate strings

The statements in a Python program are executed in sequence.

```python
# Calculates the radius of a circle.
# Author: Damir Azhar
#
import math

area = 221.67
radius = math.sqrt(area / math.pi)
print("Radius of circle with area ", area, "is", radius)
```

```python
number = 34
number = 56
number = number - 10
print("Finally", number)
```

```
num1 = 7
num2 = 3
num3 = 2
num4 = 4

num5 = num1
num1 = num2 * num1 + 4
num2 = num5 + num2
num5 = num3
num3 = num4 - num3 + 1
num4 = num5

print(num1, num2, num3, num4, num5)
```
Another Python type - strings

**Strings are any sequence of characters** enclosed inside single quotes ('...') or double quotes ("..."). We have already met strings when we needed to print a message to the standard output, e.g.,

```python
print("Area of circle")
```

Examples of strings:
- "A"
- 'A longer string'
- "45.78"
- " ">
- 

The Python `len()` function

Python has a built-in function, `len()`, which can be used to determine the length of a string.

```python
word1 = "Fantastico"
length1 = len(word1)
length2 = len("012 3 4")
print(length1, length2)
```

In the example code above there are two calls to the `len()` function (on the right hand side of lines 2 and 3).

The `len()` function is said to **return** the number of characters in the string passed to the function (inside the parentheses).

Functions use round brackets (parentheses).

On line 2 of the code, the string, `word1`, is **passed to** the `len()` function. On line 3 of the code, the string, "012 3 4", is **passed to** the `len()` function.

The `len()` function **returns** the number of characters in the string (passed to the function inside the parentheses).
In Python everything is an object

The world is made up of real world objects e.g. students, dogs, cars, cats, books, words, numbers. Objects are the things our programs deal with and in our programs we want to represent these objects.

So far, in our programs, we have used:
- **Integer** objects which represent whole numbers,
- **Floating point** objects which represent decimal numbers, and,
- **String** objects which represent sequences of characters.

We have used variables to store these types of objects in the program memory.

In Python everything is an object

We often visualise variables as being a box containing a value (the last value assigned to the variable). Given the code:

```python
box_size = 5
box_area = box_size * box_size
```

we visualise the two variables:

<table>
<thead>
<tr>
<th>box_size</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>box_area</td>
<td>25</td>
</tr>
</tbody>
</table>

In fact, every variable in Python stores a reference (the memory address) of the value assigned to it:

<table>
<thead>
<tr>
<th>box_size</th>
<th>010100101</th>
</tr>
</thead>
<tbody>
<tr>
<td>box_area</td>
<td>100001011</td>
</tr>
</tbody>
</table>

Storing the reference (the memory address) of the value assigned to a variable makes sense:

```python
initial = "A"
phrase = "The early bird catches the worm but the second mouse gets the cheese!"
phrase = "Illiterate? Write For Help"
```

because the information inside an object can have different sizes.

**Exercise**

Given the following code:

```python
item1 = "Blah!"
item2 = "Blah?"
item3 = item2
item2 = item1
```

how many string objects are there in memory?

Given the memory diagram below, i.e., fill in the variable addresses:

<table>
<thead>
<tr>
<th>item1</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Blah!&quot;</td>
<td></td>
</tr>
<tr>
<td>item2</td>
<td>111</td>
</tr>
<tr>
<td>&quot;Blah?&quot;</td>
<td></td>
</tr>
<tr>
<td>item3</td>
<td></td>
</tr>
</tbody>
</table>
None is a special value which can be assigned to a variable and it means that the variable is not referencing (pointing to) any object.

```python
initial = "A"
phrase = "The early bird catches the worm but the second mouse gets the cheese!"
phrase = None
```

A variable which contains the value None can be printed:

```python
phrase = None
print(phrase)
None
```

Special characters in a string

Some characters perform operations such as inserting a new line or a tab space. To insert a new line into a string we use the escape sequence `\n` within the string, and `\t` is used to insert a tab space.

```python
shopping = "Carrots, \npumpkin, \nchocolate"
print(shopping)
```

To insert a double quote into the output (if your string is enclosed inside double quotes), use the escape sequence `\\` , and to insert a single quote into the output, (if your string is enclosed inside single quotes), use the escape sequence `\'`.

```python
print(1, "\\"Super\\" Man")
print(2, '\"Super\' Man')
print(3, '"Super" Man')
print(4, "Super Ma\n")
```

The inbuilt type() function

Every Python object has a specific type. The type of any Python object can be obtained by using the type() function. This function returns a string stating the object type. For example

```python
num1 = 7
num2 = 26.7
word = "numero"

print(type(num1))
print(type(num2))
print(type(word))
```

The output, `<class 'int'>` means that there is the definition of this type of object in a file named int.py (inside the Python libraries)

More about strings

A string is a sequence of characters and every character in a string has an index, i.e., its position in the string. The index starts from position 0. For example:

```python
greeting = "Hello World"
```

Every character in the string can be accessed using the variable name, square brackets and the index value:

```python
greeting = "Hello World"
first_letter = greeting[0]
last_position = len(greeting) - 1
last_letter = greeting[last_position]
print(first_letter, last_letter)
```

```python
greeting = "Hello World"
first_letter = greeting[0]
last_position = len(greeting) - 1
last_letter = greeting[last_position]
print(first_letter, last_letter)
```
Ooops!

What is the problem with the following code?

```python
... 4 greeting = "Hello World"
5 last_letter = greeting[len(greeting)]
```

Traceback (most recent call last):
  File "LectureCode.py", line 5, in <module>
    last_letter = greeting[len(greeting)]
IndexError: string index out of range

An IndexError occurs if you try to access a position in the string which doesn't exist

Strings – negative index

To access a character from the end of the string, a negative index can be used. For example

```python
... greeting = "Hello World"
last_letter = greeting[-1]
second_to_last = greeting[-2]
print(last_letter, second_to_last)
```

Does the following code cause a problem?

```python
... greeting = "Hello World"
a_letter = greeting[-len(greeting)]
```

Slicing strings

As well as obtaining a single character from a string, a whole sections of the string can be obtained. This is called **slicing**.

```python
... greeting = "Hello World"
first_part = greeting[0:5]
second_part = greeting[6:]
print(second_part, first_part)
```

To get a section of a string we use square brackets, the index of the first character in the section we want, a colon followed by the index of the character after the end of the required section.

```python
... greeting = "Hello World"
first_part = greeting[0:5]
second_part = greeting[6:11]
print(second_part, first_part)
```

Slicing strings

When slicing a string, if the start of the slice is omitted, the slice starts from the first character in the string. When slicing a string, if the end of the slice is omitted, the slice goes to the end of the string.

```python
... greeting = "Hello World"
first_part = greeting[0:5]
second_part = greeting[6:]```
**Concatenation - joining strings**

The + operator can be used to join two strings, e.g.,

```python
first_name = "Alan"
last_name = "Turing"
full_name = first_name + " " + last_name
print("***", full_name , "***")
```

How does the Python interpreter know if the + operator is adding two numbers or concatenating two strings?

```python
first = "4"
second = "5"
sum = 4 + 5
number = first + second
print(sum, number)
```

**The repeat operator – repeat strings**

The * operator can be used to create a new string object with characters of a string repeated two or more times, e.g.,

```python
praise = "good!"
lots_of_praise = praise * 4
print(praise)
print(lots_of_praise)
good!
good!good!good!good!
```

What is the meaning of the words "to create a new" string object in the statement above?

**Complete the output**

Complete the output.

```python
s = "Dogs have masters. Cats have staff."
print("1.", s[1: 6])
print("2.", s[:2] * 3)
print("3.", s[-3])
print("4.", s[4] + s[1])
print("5.", s[-4:])
```

**Exercise**

Complete the following program so that it prints the name between two rows of stars. The output has three spaces on each side of the name. Your code should work for names of any length.

```python
name = "Philomena Evangeline"
extras = 3
```

```
**************************
Philomena Evangeline
**************************
```
In Python:
- variables store a reference to the object
- string objects are a sequence of characters
- the `len()` function is used to calculate how many characters are in a string
- we use the index number to obtain a single character from a string
- we can slice strings
- use the `+` operator to concatenate strings

Examples of Python features used in this lecture

```python
words = " Prince Charming ">
length = len(words)

letter1 = words[3]
letter2 = words[-5]
letter3 = words[len(words) - 2]

letters1 = words[3:6]
letters2 = words[:6]
letters3 = words[6:]
letters4 = words[len(words) - 3:]

word2 = letter1 + letter2
phrase = word2 + " " + letter3
print(letters1, letters2, letters3, letters4, phrase)
```