Programming – Using Strings

A **string** is a particular type of data used in computer programming. Strings refer to **sequences of characters** that can be found on a keyboard. These can be thought of as words or sentences.

When we write strings we use **"quote marks"** around them.

Examples of strings include:

- `"Clive"`
- `"badger"`
- `"The answer is "`
- `"&$%£#"`
- `"RedDog2017"`

Note that strings can include spaces, numbers, symbols and anything else that would usually be found on a keyboard.

1 Basic Strings

Strings are an important type of data that can be stored using variables. When users input values in Python they are always entered as strings.

Strings can be entered, stored and output. The pseudo-code commands to do this are simple:

```
OUTPUT "Enter your password"

password ← USERINPUT

OUTPUT "I know your password is"

OUTPUT password
```

Note that on the first and third lines strings are used directly in the output. In the last line the variable password is output – again as a string.

Exercise 1.1

a) write down a suitable string value to be stored in a variable called firstName

b) which of the following are valid string values:

```
  (i)   "asparagus"
 (ii)   "73.6"
(iii)   "/"
 (iv)   42
  (v)   " "
```

Note that strings can be made up just of spaces. They can also have nothing at all in them. This is called an **empty string** and is written "". This is the value that would be assigned if a user just pressed return when asked to enter a value. Empty strings contain nothing but definitely exist – and are a really useful idea of know about.

Exercise 1.2

Write a definition of a string.

Programming exercise 1.1

Try this program in Python. Try entering different values each time you run it – including symbols, numbers and just pressing return to enter an empty string.

```
# program using strings
myString = "Python Programming"
print(myString)
print() # prints an empty line
userName = input("Enter your username: ")
print(userName)
```

You may already know how to improve this program by adding text to the last line to make the output make more sense.

2 Using Strings

A string is a sequence of characters. This means it includes a number of individual characters – including spaces, symbols and numbers. The string "Blue dog" contains 8 characters – 7 letters and the space between the 'e' and the 'd'.

✎Exercise 2.1

Give the length of each of the following strings:

```
a) "banana"
b) "dancing with ponies"
c) "NitrousOxide"
d) "39.03"
e) " "
f) ""
```

There is a pseudo-code command that provides the length of any string

```
lengthOne ← LEN(myString)

lengthTwo ← LEN("Cowabunga")
```

The first command will give the length of whatever variable is stored in myString. The second will give the length of the string "Cowabunga" – which is 9.

✎Exercise 2.2

Write down the value of length returned in each of the following cases

```
a) length ← LEN("apples and pears")
b) length ← LEN("42")
c) myString ← "oranges and lemons"
   length ← LEN(myString)
```

In Python the command `len()` has the same effect as the pseudo-code command `LEN()`.

💻Programming exercise 2.1

Try this programming exercise.

```
# program to deal with length of strings
theString = input("Enter a string: ")
length = len(theString)
print(length)
```

Try it with different values, including symbols, numbers, spaces and by entering an empty string.

3 Working Inside Strings

Strings are sequences of characters – such as "a", "n" and "d" which make the string "and". Each character in the sequence can be identified individually. To do this we need to know the position within the a string that each character is located at.

Strings are indexed. This means that each position within the string is given a numeric value called an **index**. Indexes start from 0 – which means that the first character in a string is at index 0 (position 0).

If myString has a value of "squirrel" then each character has an index as shown:

index	0	1	2	3	4	5	6	7
value	s	q	u	i	r	r	e	l

The q in "squirrel" is at index 1, the e at index 6 and the s at index 0.

🖉Exercise 3.1

The value of myString is `"fox and goose"`.

a) what is the value of LEN(myString)

b) what value is stored at each of the following indexes:

(i) 3
(ii) 5
(iii) 1
(iv) 12
(v) 0
(vi) 13

c) what is the index of the value "d" within myString?

Starting indexes from 0 can get slightly confusing at times, but it's something that can also come in useful. Most programming languages start the indexes of strings from 0 so it's just something you have accept.

Finding a character within a string

It is possible to find the index value of a character within a string. This will give the first occurrence of the character within the string.

In pseudo-code this is done using the command POSITION.

```
placeInString ← POSITION("apples", "p")
```

The first time 'p' occurs in the string "apples" is at index 1 – the second character in the string. So this will assign the value 1 to placeInString.

✎ Exercise 3.2

if myString has the value "january", what value will be returned in each case:

```
a) POSITION(myString, "n")
b) POSITION(myString, "j")
c) POSITION(myString, "a")
```

💻 Programming exercise 3.1

In Python the POSITION command is implemented using the command `find()`.

Try this program.

```
# program to use find in a string
myString = "december"
placeC = find(myString, "c")
print(placeC)
placeD = find(myString, "d")
print(placeD)
placeE = find(myString, "e")
print(placeE)
```

Now try adding this to the program

```
# a value that's not in the string
placeY = find(myString, "y")
print(placeY)
```

Accessing characters in a strong

In Python we can access each character in a string by using a set of square brackets:

```
myChar = myString[3] # finds the value at index 3
```

💻Programming exercise 3.2

Try this out.

```
myString = "squirrel"
print(myString[3])
print(myString[0])
print(myString[7])
```

💻Programming exercise 3.3

Now try adding this to the previous program.

```
print(myString[8]) # produces an error
```

The error occurs because there is no index 8 in myString – it only goes to index 7. This produces an "index out of range" error. Whenever you see this you know that you're looking for something in a sequence that isn't actually there.

---

4 Working with part of a string:

It is sometimes useful to get the value of only part of a string. This uses an idea called a **substring**.

A substring is part of a string. So the substring "and" is part of the longer string "fox and goose". It is made up of the values between index 4 and index 6 of the longer string.

In pseudo-code there is a command which easily lets us create a substring from a string.

```
shortString ← SUBSTRING(4, 6, "fox and goose")
```

The command uses three parameters. The 4 and the 6 tell you which values to use as the start and end of the substring, and the last parameter is the longer string to create the substring from.

```
shortString ← SUBSTRING(0, 6, myString)
```

This would create a new string made up of the first 7 characters (from index 0 to index 6) of the value stored in myString. If myString had the value "computer science" the value of shortString would be "compute"

✎Exercise 4.1

a) If myString has the value "queen anne" what substring would be returned in each of these cases:

```
 (i)   SUBSTRING(2, 4, myString)
(ii)   SUBSTRING(7, 9, myString)
(iii)  SUBSTRING(0, 4, myString)
(iv)   SUBSTRING(0, 7, myString)
```

b) What value would be assigned to shortString at the end of this pseudo-code:

```
myString ← "acacia avenue"
shortString ← SUBSTRING(1, 7, myString)
```

Substrings are a little more complex in Python and use an idea called **slicing**.

```
myString = "squirrel"
partString = myString[2:4] # gives the value "ui"
```

- The 2 represents the index to start from – the u is at index 2
- The 4 represents the index to stop at – but you stop **before** you add it to the substring. This is quite confusing – and different to using pseudo-code.

💻Programming exercise 4.1

```
# creating substrings
myString = "squirrel"
part1 = myString[2:4]
print(part1)
part2 = myString[2:3]
print(part2)
part3 = myString[0:6]
print(part3)
```

This should print “ui”, “u” and “squirr”

💻Programming exercise 4.2

Now try:

```
part4 = myString[2:2] # starts at index 2 but ends before index 2
print(part4)
```

💻Programming exercise 4.3

And now try:

```
part5 = myString[4:8]
print(part5)
part6 = myString[4:9] # index 9 is beyond the end of the string
print(part6)
```

To make this easiy to avoid you can be clever and use the len function:

```
part7 = myString[4:len(myString)]
print(part7)
```

💻Programming exercise 4.4

You can use slicing to get the start or end section of a string as well:

```
myString = "squirrel"
start = myString[:2] # from the beginning
end = myString[4:] # to the end
print(start)
print(end)
```

5 Adding strings together

You can combine strings to form a longer string. This simply uses the + operator and is a process called **concatenation**.

```
longString ← "badger" + "fox"

OUTPUT longString
```

This combines the two strings to give "badgerfox" as the output. Note that there is no space between the two parts of the string.

🖉Exercise 5.1

What value is output at the end of each of these sets of pseudo-code.

a)

```
stringOne ← "apples"
longString ← stringOne + "pears"
OUTPUT longString
```

b)

```
stringOne ← "oranges"
stringTwo ← "lemons"
longString ← stringOne + stringTwo
OUTPUT longString
```

c)

```
stringOne ← "fish"
stringTwo ← "chips"
longString ← stringOne + " and " + stringTwo
OUTPUT longString
```

Concatenation works exactly the same way in Python,

💻Programming exercise 5.1

Try this program.

```
# program using concatenation
stringOne = "punch"
stringTwo = "judy"
longString = stringOne + stringTwo
print(longString)
```

Improve the program to add the word "and" and spaces between each section of longString.

💻Programming exercise 5.2

Write a program to allow a user to enter two strings. The program should then concentrate them together using an "and" between the two parts.

💻Programming exercise 5.3

Concatenation can be used with substring techniques as well

```
# concatenation using substrings
myString = "squirrel"
start = myString[:1]
partString = myString[3:6]
theString = start + partString
print(theString)
```

Summary

Strings are sequences of characters. They are **indexed** from position 0. The **length** of a string and the **position** of a character within a string can be found. Strings can be split into **substrings** and can be joined together using **concatenation**.

Solutions to exercises

Exercise 1.1:
a) You probably wrote down someone's first name

b) All apart from (iv) are suitable values. 42 is not a string – it doen't have quote marks around it. "73.6" is a string because it has quotes – numbers can be stored as words so that's OK. (v) is just a string with a space in it. That's fine as well.

Exercise 1.2:
A string is a sequence of characters

Exercise 2.1:
(i) 6
(ii) 19 – don't forget to count the spaces
(iii) 12
(iv) 5 – don't forget the dot
(v) 1 – a single space
(vi) 0 – this is an empty string

Exercise 2.2:
a) 16 – don't forget the spaces
b) 2
c) 18

Exercise 3.1
a) 13

b) Don't forget that the index starts from 0

(i) "n"
(ii) " " – a space is the character at index 3
(iii) "o"
(iv) "e" – the final letter of goose is at index 12
(v) "f"
(vi) there is no value at index 13. The index values in the string go from 0 to 12.

c) 6

Exercise 3.2:
a) 2 – the n is at index 2 – the third character in the string is index 2 because the index starts from 0
b) 0
c) 1 – the first occurrence of a is at index 1. It also occurs at index 4.

Exercise 4.1:
a)

(i) "een" – index 2 through index 4
(ii) "nne" – don't forget to count the space
(iii) "queen"
(iv) "queen an"

b) "cacia a"

Exercise 5.1
a) "applespears"
b) "orangeslemons"
c) "fish and chips"