Learning Aims

- Be able to write programs that manipulate strings including length, position, substrings and case conversion.
- How to format a string.
- Design algorithms and write code to validate input
- Use a range of validation checks in programs
- Use built-in subprograms to manipulate variables of data type string

Concatenate	When two or more strings are joined together.
String	A value that is text. This can include numbers but they will be read as text.
String handling	Performing operations on string.
Element	A character in a string, or an item in a sequence.
ASCII	Acronym for American Standard Code for Information Interchange. It is used to represent characters with a numerical value.
Substring	Part of a string.

String Manipulation

A string is anything enclosed in quote marks. “”

Programming languages have built-in functions to manipulate strings.

The functions below are given in a Python format.

Each programming language will have its own particular functions and syntax.

Useful stings functions

To get the length of a string:

len(stringname)

To count the specific number of characters in a a string:

string.count(“letter”)

Position

Individual letters are obtained from a string by using their index.

string[index]

0	1	2	3	4
H	e	l	l	o

For example:

```
message = "Hello"
print(message[1])
```

This will output the letter ‘e’.

Python string indexes start at position 0, not 1.

This means that stringname[0] contains the first character of a string called stringname.

Substrings

A substring is part of a string. In Python, extracting a substring is carried out using string slicing.

To find a substring in Python use the syntax:

```
string[start:end]
```

For example:

```
message = "Hello everyone"
substring = message[4:8]
print(substring)
```

The output will be "o ev".

This is because position 4 in the string contains "o" and positions 5, 6 and 7 contain " ev".

Remember space is included as a position in the string.

End and start position

message = "Hello everyone"

The end position isn't included in Python.

To get the first characters from a string use:

```
string[:end]
```

For example, message[:8] will result in "Hello ev".

To get the last characters from a string use:

```
string[start:]
```

For example, message[start:]

message[11:] will result in the substring "one".

Substring position

It is possible to find the position of a substring by using string.find(substring)

The position will be returned. If the substring isn't in the string then -1 will be returned.

For example:

```
message = "Hello everyone"
position = message.find("very")
print(position)
```

This will return the position 7 as the 'v' in "very" is in index 7 in the string.

Summary

In the statements below, the result of each operation is shown in a comment on the right.

```
myName = "John Robinson”
firstname = myName[0:4]                          # "John"
surname = myName[5:]                             # "Robinson"
numChars = len(myName)                           # 13
lastTwoChars = myName[len(myName)-3:]            # "on"
lastTwoChars = myName[-2:]                       # "on"
positionOfRob = myName.find("Rob")               # 5
```

Case conversion

To convert cases:

```
string.upper()
string.lower()
```

For example:

```
message = "Hello everyone"
print(message.lower())
print(message.upper())
```

This will return:

```
hello everyone
HELLO EVERYONE
```

ASCII characters

Every character is represented in binary, for example using the ASCII representation. ASCII, “A” is
represented by the same binary code as the decimal number 65, “B” is 66, “C” is 67,… “Z” is 90.

The functions ord and chr convert characters between ASCII and decimal.

```
num = ord("A") # makes num = 65
letter = chr(66) # makes letter = "B"
```

Menus

- The user input can be validated using a range check to ensure that the option selected is allowed.

```
******Menu******

1. Display my name
2. Display my age
3. Display my address

What is your menu option?
```

```
1  # -----------------------------------------------------------
2  # Global variables
3  # -----------------------------------------------------------
4  validChoice = False             # Assume everything is invalid
5  userChoice = 0                  # Set to an invalid value
6
7  # -----------------------------------------------------------
8  # Subprograms
9  # -----------------------------------------------------------
10 def showMenu():
11     print ("Option 1: Find the highest value")
12     print ("Option 2: Find the lowest value")
13     print ("Option 3: Calculate the average")
14
15 # -----------------------------------------------------------
16 # Main program
17 # -----------------------------------------------------------
18
19 while (not validChoice):
20     showMenu()
21     userChoice = int (input ("Enter an option: "))
22     if (userChoice >= 1) and (userChoice <= 3):
23         validChoice = True
24     else:
25         print ("Invalid option, try again ")
26
27 print ("You entered option " + str(userChoice))
```

Validating input data

Input validation is when a system will check that the input meets certain criteria, to ensure that the data is in an acceptable form.

- **Range check**: a number or date is within a sensible/allowed range
- **Type check**: data is the right type such as an integer, a letter or text
- **Length check**: text entered is not too long or too short – for example, a password is greater than 8 characters, a product description is no longer than 25 characters
- **Presence check**: checks that some data has been entered, i.e. that the field has not been left blank
- **Pattern check**: checks that the pattern of, for example, a postcode or email address is appropriate

Other useful string functions

The following string functions are also available in Python. You should be aware of how these work as they are included in the Edexcel PLS

String function	Description	Example

`string.isalpha()`	Checks if the string contains alphabetic letters. If it does then it returns True.	`message = "Southampton"`
		`print(message.isalpha())`
		`Output: True`

`string.isalnum()`	Checks if the string contains alphanumeric letters and numbers A-Z and 0-9. Returns True if that's all it contains.	`message = "Southampton23"`
		`print(message.isalnum())`
		`Output: True`

`string.isdigit()`	Returns true if the string contains just digits 0-9.	`message = "498"`
		`print(message.isdigit())`
		`Output: True`

`string.replace(string1, string2)`	Replaces all occurrences of string1 with string2.	`message = "Southampton"`
		`print(message.replace("South", "North"))`
		`Output: Northampton`

`string.split(character)`	Splits a string on each occurrence of the character. Each substring is stored in a list.	`message = "Lemon,Apple,Peach"`
		`fruitList = message.split(",")`
		`print(fruitList)`
		`Output: ["Lemon", "Apple", "Peach"]`

`string.strip(character)`	Remove all occurrences of the character from the string.	`message = "...Beach.."`
		`print(message.strip("."))`
		`Output: Beach`

`string.isupper()`	If all the characters in the string are in uppercase then it returns True.	`message = "HELLO"`
		`print(message.isupper())`
		`Output: True`

`string.islower()`	If all the characters in the string are in lowercase then it returns False.	`message = "hello"`
		`print(message.islower())`
		`Output: True`

`string.index(substring)`	This works in the same way as `string.find(substring)` which was discussed earlier. The difference is that it raises an exception (rather than returning -1) if the substring isn't found. Use this only if you are using exceptions.	

Validation Routines and While Loops

When we attempt to introduce input validations to our program, we will often make use of the while loop construct.

Here is an example of a simple validation routine that ensures that the program only accepts the inputs YES and NO, rejecting all other inputs:

```
Do you wish to carry on? Type YES or NO
yessss
Wrong word entered, please type YES or NO
yes
Wrong word entered, please type YES or NO
YES
You want to carry on!
```

Validation Routines and While Loops

A while loop to repeatedly ask for an input until the user enters YES or NO. This is the validation routine.

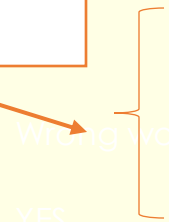
An input statement to record the user's answer to the question

```
answer = input("Do you wish to carry on? Type YES or NO")


while (answer != "YES") and (answer != "NO"):
    answer = input("Wrong word entered, please type YES or NO")


if answer == "YES":
    print("You want to carry on!")
else:
    print("You don't want to carry on!")
```

If-Else statements to act on the users input



Range check

- The following algorithm asks the user to enter an integer between 17 and 30, and validates it by performing a range check. The number is then multiplied by 3 and the result printed out.

```
num = int(input("Enter number between 17 and 30:"))
while num < 17 or num > 30:
        num = int(input("Invalid number - please re-enter: "))
        print(num * 3)
```

Type check

The following program asks the user to enter a number. It performs a type check. If the user has entered text rather than a number, they will be asked to enter the number again.

```
num = input("Please enter an integer: ")
while not num.isdigit():
    num = input("You must enter an integer, try again: ")
    num = int(num)
print(num)
```

Other useful functions for checking the type of data entered include:

- isalpha() Returns True if all the characters are alphabetic (A-Z)
- isalnum() Returns True if all the characters are alphabetic (A-Z) or digits (0-9).

Length check

The following program asks the user to enter their name, and performs a length check.

The name must be between 2 and 20 characters.

```
name = input("Please enter name: ")
while len(name) < 2 or len(name) > 20:
        name = input("Must be between 2 and 20 characters – please re-enter: ")
print(name)
```

Presence check

- This simply ensures that a value has been presented to the program, preventing the from leaving an input blank.
- This algoirthm asks the user to input their name and uses a presence check to ensure the they have entered a value.
- This example illustrates how to do a presence check on user input from the keyboard.

```
1  # ------------------------------------------------------------
2  # Global variables
3  # ------------------------------------------------------------
4  userName = ""                # Initialise to empty string
5
6  # ------------------------------------------------------------
7  # Main program
8  # ------------------------------------------------------------
9  userName = input ("Enter your name: ")
10 while (userName == ""):
11     userName = input ("Enter your name: ")
12 print ("Your name is: ", userName)
```

Pattern check

A pattern check (also known as a format check) allows the checking of an email address, for example.

Look at the following program which makes use of the split function to check that an @ symbol is in the address.

```
validEmail = False
email = ""
while not validEmail:
email = input("Enter email address: ")
emailParts = email.split("@")
if len(emailParts) == 2:
    validEmail = True
else:
     print("That isn't a valid email address")
```

The program splits the email address entered with each occurrence of the @ symbol. This should result in a list with two items, the left hand side and right hand side of the email address.

One problem with this program is that the left hand side and right hand side of the @ symbol could be empty, and this would still result in the program choosing it as a valid email.
Adapt the program so that it checks to see there are characters before and after the @ symbol.

```
validEmail = False
email = ""
while not validEmail:
    email = input("Enter email address: ")
    emailParts = email.split("@")
    if (len(emailParts) == 2 and len(emailParts[0]) > 0
                             and len(emailParts[1]) > 0):
        validEmail = True
    else:
        print("That isn't a valid email address")
```

Validating strings

Function	Description
`len(<string>)`	Returns the length of <string>
`<str>.upper()`	Returns the original string in upper case
`<str>.lower()`	Returns the original string in lower case
`<str>.isalnum()`	Returns True, if all characters are alphabetic (a-z, A-Z,) and digits (0-9)
`<str>.isalpha()`	Returns True, if all characters are alphabetic (a-z, A-Z)
`<str>.isdigit()`	Returns True, if all characters are digits (0-9)

Patterns with only letters

- A program may be written to accept a person's first name. The programmer makes a set of rules for validating first name. The rules are:
- must have at least 1 letter and no more than 20 letters
- must only have letters, i.e. no digits or symbols
- This example shows how to implement the two rules described above.

Using isalpha() to check is all the input only have letters

```
# -----------------------------------------------------------
# Constants
# -----------------------------------------------------------
MIN_NAME = 1                # Constants for name length
MAX_NAME = 20

# -----------------------------------------------------------
# Global variables
# -----------------------------------------------------------
firstName = ""              # User types
valid = False               # Assume everything is invalid
length = 0                  # Invalid length
newName = ""                # Invalid capitalised name

# -----------------------------------------------------------
# Main program
# -----------------------------------------------------------
while (not valid):
    firstName = input ("Enter your first name: ")
    length = len(firstName)
    if (length >= MIN_NAME) and (length <= MAX_NAME):
        if (firstName.isalpha()):
            newName = firstName
            print ("Your entry is valid.\nYour name is: ", newName)
            valid = True
        else:
            print ("Your entry is not valid")
    else:
        print ("Your entry is not the right size")
```

Patterns with both letters and digits

- Identification numbers, stock identification and postcodes are all items that may be combination of both letters and digits. These can be validated by the built-in subprograms.

- This example validates the strings representing form names. Examples of valid form names are "7AXB", "8PBD","9ARL". From this we create these rules:

  - must start with the digits 7,8,9
  - must be followed by three letters, but we can adjust the case
  - must be exactly four characters long

```
# ---------------------------------------------------------
# Constants
# ---------------------------------------------------------
FORM_LEN = 4            # Constants for name length

# ---------------------------------------------------------
# Global variables
# ---------------------------------------------------------
formName = ""               # User types
valid = False               # Assume everything is invalid

# ---------------------------------------------------------
# Main program
# ---------------------------------------------------------
while (not valid):
    formName = input ("Enter a form name: ")
    if (len(formName) == FORM_LEN):        # Length is good
        if (formName.isalnum()):   # Letters and digits only
            # Check first character is 7, 8, or 9
            if ((formName[0] == "7") or
                (formName[0] == "8") or
                (formName[0] == "9")):
                # Good first char, check remaining three
                if (formName[1].isalpha() and
                    formName[2].isalpha() and
                    formName[3].isalpha()):
                    valid = True
                    print ("Your entry is valid.", formName)
                else:
                    print ("Last three characters must be letters")
            else:
                print ("First character must be 7, 8, or 9")
        else:
            print ("Your entry is not valid")
    else:
        print ("Form name is not correct length")
```