6 Strings and Text Files

Learning Outcome

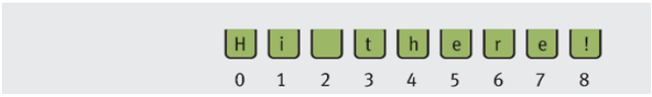
Programming Elements and Consturcts
Understand the data type string and use common library functions for strings
Implementing Algorithms and Data Structures
Store data in and retrieve data from serial and sequential text files

6.1 Accessing Characters and Substrings in Strings

6.1.1 The Structure of Strings

- Data structure: Consists of smaller pieces of data
- String's length: Number of characters it contains (0+)

```
>>> len("Hi there!")
9
>>> len("")
0
```



6.1.2 The Subscript Operator

The form of the **subscript operator** is:

```
<a string>[<an integer expression>]
```

index is usually in range [0,len); can be negative

```
>>> name = "Alan Turing"
>>> name[0]                      # Examine the first character
'A'
>>> name[3]                      # Examine the fourth character
'n'
>>> name[len(name)]              # Oops! An index error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name[len(name) - 1]          # Examine the last character
'g'
>>> name[-1]                     # Shorthand for the last one
'g'
```

Subscript operator is useful when you want to use the positions as well as the characters in a string. For example, we can use a count-controlled loop below.

```
>>> data = "Hi there!"
>>> for index in range(len(data)):
        print(index, data[index])

0 H
1 i
2
3 t
4 h
5 e
6 r
7 e
8 !
>>>
```

6.1.3 Slicing for Substrings

Python's subscript operator can be used to obtain a substring through a process called **slicing**. Place a colon (:) in the subscript; an integer value can appear on either side of the colon.

```
>>> name = "myfile.txt"
>>> name[0:]               # The entire string
'myfile.txt'
>>> name[0:1]              # The first character
'm'
>>> name[0:2]              # The first two characters
'my'
>>> name[:len(name)]       # The entire string
'myfile.txt'
>>> name[-3:]              # The last three characters
'txt'
```

6.1.4 Testing for a Substring with the in Operator

When used with strings, the left operand of **in** is a target substring and the right operand is the string to be searched. Returns **True** if target string is somewhere in search string, or **False** otherwise.

```
>>> fileList = ["myfile.txt", "myprogram.exe", "yourfile.txt"]
>>> for fileName in fileList:
       if ".txt" in fileName:
          print(fileName)

myfile.txt
yourfile.txt
>>>
```

6.2 String Methods

Python includes a set of string operations called **methods** that make tasks like counting the words in a single sentence easy. A method behaves like a function, but has a slightly different syntax.

A method is always called with a given data value called an object.

```
<an object>.<method name>(<argument-1>, …, <argument-n>)
```

- Methods can expect arguments and return values
- A method knows about the internal state of the object with which it is called
- In Python, all data values are in fact objects

STRING METHOD	WHAT IT DOES
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.

STRING METHOD	WHAT IT DOES
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.

```
>>> s = "Hi there!"
>>> len(s)
9
>>> s.center(11)
' Hi there! '
>>> s.count('e')
2
>>> s.endswith("there!")
True
>>> s.startswith("Hi")
True
>>> s.find('the')
3
>>> s.isalpha()
False
>>> 'abc'.isalpha()
True
>>> "326".isdigit()
True
```

STRING METHOD	WHAT IT DOES
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.upper()`	Returns a copy of `s` converted to uppercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.

```
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> "".join(words)
'Hithere!'
>>> " ".join(words)
'Hi there!'
>>> s.lower()
'hi there!'
>>> s.upper()
'HI THERE!'
>>> s.replace('i', 'o')
'Ho there!'
>>> " Hi there! ".strip()
'Hi there!'
```

Example: counting number of words

```
>>> sentence = input("Enter a sentence: ")
Enter a sentence: This sentence has no long words.
>>> listOfWords = sentence.split()
>>> print("There are", len(listOfWords), "words.")
There are 6 words.
>>> sum = 0
>>> for word in listOfWords:
        sum += len(word)

>>> print("The average word length is", sum / len(listOfWords))
The average word length is 4.5
```

Example: extracting a filename's extension

```
>>> "myfile.txt".split(".")
['myfile', 'txt']
>>> "myfile.py".split(".")
['myfile', 'py']
>>> "myfile.html".split(".")
['myfile', 'html']
```

The subscript [-1] extracts the last element. It can be used to write a general expression for obtaining any filename's extension, as follows:

```
filename.split(".")[-1]
```

A string is an **immutable data structure**. When we use any method, it does not change the string itself. Note the difference of the following two sets of code.

```
>>> s = 'sample'
>>> s.upper()
'SAMPLE'
>>> s
'sample'
```

```
>>> s = 'sample'
>>> s = s.upper()
>>> s
'SAMPLE'
```

6.3 Text Files

A text file is software object that stores data on permanent medium such as disk or CD. When compared to keyboard input from human user, the main advantages of taking input data from a file are:

- The data set can be much larger
- The data can be input much more quickly and with less chance of error
- The data can be used repeatedly with the same program or with different programs

6.3.1 Text Files and Their Format

Using a text editor such as Notepad or TextEdit, you can create, view, and save data in a text file. All data output to or input from a text file must be **strings**.

METHOD	WHAT IT DOES
`open(pathname, mode)`	Opens a file at the given pathname and returns a **file** object. The **mode** can be `'r'`, `'w'`, `'rw'`, or `'a'`. The last two values, `'rw'` and `'a'`, mean read/write and append, respectively.
`f.close()`	Closes an output file. Not needed for input files.
`f.write(aString)`	Outputs **aString** to a file.
`f.read()`	Inputs the contents of a file and returns them as a single string. Returns `''` if the end of file is reached.
`f.readline()`	Inputs a line of text and returns it as a string, including the newline. Returns `''` if the end of file is reached.

6.3.2 Writing Text to a File

Data can be output to a text file using a **file** object. To **open** a file for output:

```
>>> f = open("myfile.txt", 'w')
```

If file does not exist, it is created in the same folder of the Python scripts.
If it already exists, Python opens it; when data are written to the file and the file is closed, any data previously existing in the file are erased.

```
>>> f.write("First line.\nSecond line.\n")
>>> f.close()
```

Alternatively, we can use the following Python codes where the file is automatically closed. (Not recommended. A-level requires a comment that file is closed.)

```
with open("myfile.txt", 'w') as f:
     f.write("First line.\nSecond line.\n")
# the file will close automatically.
```

6.3.3 Writing Numbers to a File

The **file** method **write** expects a **string** as an argument. Other types of data must first be converted to strings before being written to output file (e.g., using str).

```
import random
f = open("integers.txt", 'w')
for count in range(500):
    number = random.randint(1, 500)
    f.write(str(number) + "\n")
f.close()
```

6.3.4 Reading Text from a File

You open a file for input in a manner similar to opening a file for output. If the pathname is not accessible from the current working directory, Python raises an error.

```
>>> f = open("myfile.txt", 'r')
```

- the **read** method

```
>>> text = f.read()
>>> text
'First line.\nSecond line.\n'
>>> print text
First line.
Second line.
```

- the **readline** method

```
f = open('myfile.txt','w')
f.write('First line.\nSecond line.\n')

f = open('myfile.txt','r')
line = f.readline()

while line != '':
      print(line)
      line = f.readline()
```

6.3.5 Reading Numbers from a File

Text file "Integers1.txt" 10 20 40 60	Text file "Integers2.txt" 10 20 40 60

`f = open("Integers1.txt",'r')`
`sum = 0`
 	
`for line in f:`
`  line = line.strip()`
`  number = int(line)`
`  sum += number`
 	
`print("The sum is ",sum)`
`f.close()`
	`f = open("Integers2.txt",'r')`
	`sum = 0`
	
	`for line in f:`
	`  wordlist = line.split()`
	`  for word in wordlist:`
	`  number = int(word)`
	`  sum += number`
	
	`print("The sum is ",sum)`
	`f.close()`

`with open("Integers1.txt", 'r') as f:`
`  sum = 0`
`  for line in f:`
`  line = line.strip()`
`  number = int(line)`
`  sum += number`
`print("The sum is ",sum)`
	`with open("Integers1.txt", 'r') as f:`
	`  sum = 0`
	`  for line in f:`
	`  wordlist = line.split()`
	`  for word in wordlist:`
	`  number = int(word)`
	`  sum += number`
	`print("The sum is ",sum)`

Notice the different ways of reading numbers based on the format of the text file. The outcome for both codes is

The sum is 130

Tutorial 6

1. Write in pseudocode, an algorithm to accomplish the following:
   (a) Replaces s1 with s2.
   (b) Appends s2 to s1.

2. Assume that the variable **data** refers to the string "**myprogram.exe**". Write the values of the following expressions:
   (a) data[2]
   (b) data[-1]
   (c) len(data)
   (d) data[0:8]

3. Assume that the variable **data** refers to the string "**myprogram.exe**". Write the expression that perform the following tasks:
   (a) Extract the substring "**gram**" from **data**.
   (b) Truncate the extension "**.exe**" from **data**.
   (c) Extract the character at the middle position from **data**.

4. Assume that the variable **data** refers to the string "**Python rules!**" . Use a string method to perform the following tasks:
   (a) Obtain a list of words in the string
   (b) Convert the string to uppercase
   (c) Locate the position of the string "**rules**".
   (d) Replace the exclamation point with a question mark.

5. Assume that the variable **data** refers to the string "**Python rules!**" . Write the values of the following expressions:
   (a) data.endswith('i')
   (b) " totally ".join(data.split())

6. Write a code segment that opens a file named **myfile.txt** for input and prints the number of lines in the file.

7. Assume that a file named **word.txt** contains words separated by newlines. Write a code segment that opens a file for input and prints the number of four-letter words in the file.

8. Assume that a file named **integer.txt** contains integers separated by newlines. Write a code segment that opens the file and prints the average value of the integers.

9. How many different codes can you think about to read a text file?

Assignment 6

1. Assume that the variable **myString** refers to a string. Write a code segment that uses a loop to print the characters of the string in reverse order.

2. Write a function that takes a binary number of up to 16 digits, and converts it into a hexadecimal number.

3. Write a script named **copyfile.py**. This script should prompt the user for the names of two text files. The contents of the first file should be input and written to the second file. You may use the text files in Tutorial 6 question 6 and 7 for testing purpose.

4. The Payroll Department keeps a list of employee information for each pay period in a text file. The format of each line of the file is the following:

   <last name> <hourly wage> <hours worked>

   Write a program that reads data from the file Payroll.txt and prints to the terminal a report of the wages paid to the employees for the given period. The report should be in tabular format with the appropriate header. Each line should contain an employee's name, the hours worked and the wages paid for that period.