Fundamentals of Python: First Programs

Chapter 4: Strings (Indexing, Slicing, and Methods)

Objectives

After completing this lesson, you will be able to:

1) Know the definition of a string and that strings are "immutable objects",
2) Access individual characters in a string,
3) Retrieve a substring from a string,
4) Search for a substring in a string, and
5) Use string methods to manipulate strings.

Problem Statement

- Ask the user for their full name: first name, one space, middle name, one space, and last name, then press return. Your job is to separate this full name into three separate variables: one for the first name, one for the middle name, and a third for the last name. Display these separate names on the screen. Also display the last name, a comma, a space, and the first name. You will need to use Python's string manipulation features.
- Write a BRIEF algorithm in pseudo code indicating how you would **separate** the names.

Algorithm Discussion

- BRIEFLY discuss the essentials of the algorithm with your row partner.
  - Start with a brief private thinking time
  - We will use “Listen & Compare”
  - One group share your algorithm.

Today's Lesson "The How"

- Reading a "Technical Text"
- Determine the central ideas of the text, summarizing the complex concepts, processes, and/or information presented in the text by paraphrasing them in simpler but still accurate terms.
- Determine the meaning of symbols, key terms, and Python commands.

Today's Lesson "The What"

- Strings: Definition, Indexing, Slicing, and String Methods
- Read Section 4.1 (Pages 122 – 125) AND Section 4.4 (Pages 136 – 140)

Today's Lesson "The How Part 2"

- Start with private thinking time.
- We will use "Listen & Compare" Structured Discussion with your partner.
- Groups will share (Explain to your partner)
  - What you learned including:
  - The Definition of a String
  - Characteristic (property – vocabulary word) of a String
  - How to use the string command/method and
  - what type of problem could you use this command.
    - Try it out in IDLE if you wish

Algorithm Discussion Part 2

- List as many ways as possible to find the spaces and/or separate the names.

- If we wanted to “error trap” the Names program,
  - which methods would we use and **why** would you use them?
  - What order would you use the commands/methods?
    - What about the “in” Operator?

Exit Ticket: Short Quiz

- Socrative.com
- Room number: LCHS607

- If time permits, start program 4A and 4B

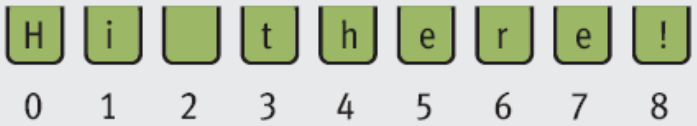
Accessing Characters and Substrings in Strings

- In this section, we examine the internal structure of a string more closely
- You will learn how to extract portions of a string called **substrings**

The Structure of Strings

- An integer can't be factored into more primitive parts
- A string is an **immutable data structure**
  - Data structure: Consists of smaller pieces of data
  - String's length: Number of characters it contains (0+)

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



[FIGURE 4.1] Characters and their positions in a string

The Subscript Operator

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

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

index is usually in range [0,length of string – 1]; can be negative

- Examples:

```
>>> 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'
```

The Subscript Operator (continued)

- Subscript operator is useful when you want to use the positions as well as the characters in a string
  - Use a count-controlled loop

```
>>> 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 !
>>>
```

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'
```

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
>>>
```

String Methods

- Python includes a set of string operations called **methods** that make tasks like counting the words in a single sentence easy

```
>>> 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
>>>
```

String Methods (continued)

- 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 objects

String Methods (continued)

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.
`s.endswith(sub)`	Returns `True` if `s` ends 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.

[TABLE 4.2] Some useful string methods, with the code letter `s` used to refer to any string

String Methods (continued)

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.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.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`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.
`s.upper()`	Returns a copy of `s` converted to uppercase.

[TABLE 4.2] Some useful string methods, with the code letter `s` used to refer to any string

String Methods (continued)

```
>>> 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
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> "".join(words)
'Hithere!'
```

String Methods (continued)

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

String Methods (continued)

- 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
  - Can be used to write a general expression for obtaining any filename's extension, as follows:

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

Summary

- A string is a sequence of zero or more characters
  - Immutable data structure
  - `[]` used to access a character at a given position
    - Can also be used for **slicing** (`[<start>:<end>]`)
- **`in`** operator is used to detect the presence or absence of a substring in a string
- Method: operation that is used with an object
- The string type includes many useful methods for use with string objects