



Strings in Python



At the end of this lesson, you should be able to:

- Describe the terminology, 'string', as used in Python
- Define strings in Python
- Access characters in a Python string
- Slice a Python string to get substrings
- Use common string functions and methods in Python

Topic Outline



Definition of a String



Accessing Strings



Strings Operations



Functions and Methods for String

What is a String?

A **string** is a **sequence** of characters.



Notionally, a character is a letter or a symbol.



- A string is indicated using **single quote** ('...') or **double quotes** ("...")
 - For example, 'abc' or "abc"
- The **sequence** of characters is important and is maintained

Defining a String (Cont'd)

- Using a single or double quotes is fine



Example:

`s = "a string"`

or

`s = 'a string'`

- Don't use both



Example:

`s = "a string'` → **WRONG**



Syntax Error

How to put an apostrophe in a string, e.g., Mike's book?

- `s = "Mike's book"` → Use different quotes
- `s = 'Mike\'s book'` → Use an escape character

- Characters in a string are in a **sequence**.
- We can identify each character with an unique **index** (a position in the sequence).
- We can index a character from either end of the sequence.
 - Non-negative values: counting from **left**, **starting at 0**
 - Negative values: counting from **right**, **starting at -1**

Index (Cont'd)

String = "Hello World"

| | | | | | | | | | | | |
|------------|---|---|---|---|---|---|---|---|-----|----|----|
| Characters | H | e | l | l | o | | W | o | r | l | d |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | ... | -2 | -1 |

*Indices 10 and -1 point to the same location: **d***

Accessing One Element

We can use `[]` to access particular characters in a string.

```
myStr = "Hello World"
```

```
x = myStr[1]
```

```
print(x)
```



#will print **e**

```
print(myStr)
```



? #will print **Hello World**

```
print(myStr[-2])
```



? #will print **l**

```
print(myStr[11])
```



?  Error



Characters

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|-----|----|----|
| H | e | l | l | o | | W | o | r | l | d |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | ... | -2 | -1 |

Indices

Slicing: Parts of a String

We can also select a part of the string.

Syntax: [**start** : **finish**]

↑
the index of the end of a subsequence (*not included*)

↓
the index of the start of a subsequence

*By default, these indices, (**start** and **finish**), will point to the **beginning** and **end** of the string, respectively.*

Slicing: Parts of a String (Cont'd)

The start index is inclusive while the **finish index** is the **one after** the subsequence.

```
myStr = "Hello World"
```

```
myStr[1:4] → ?
```

Characters

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|-----|----|----|
| H | e | l | l | o | | W | o | r | l | d |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | ... | -2 | -1 |

Indices

Slicing: Parts of the String (Cont'd)

Example:

```
myStr = "Hello World"
print(myStr[1:6])
print(myStr[1:2])
print(myStr[-7:-1])
print(myStr[-3:-5])
print(myStr[:6])
print(myStr[5:])
```

Output

'ello '

'e'

'o Worl'

'' (empty)

'Hello '

' World'

Characters

Indices

| | | | | | | | | | | |
|-----|-----|----|----|----|----|----|----|----|----|----|
| H | e | l | l | o | | W | o | r | l | d |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| -12 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

Slicing: Parts of the String (Cont'd)

One More Example:

| | | | | | | | | | | | | |
|------------|---|---|---|--------------------------|---|---|---|---|---|-----|----|----|
| | | | | <code>myStr[3:-2]</code> | | | | | | | | |
| Characters | H | e | l | l | o | | W | o | r | l | d | |
| Indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
| | | | | | | | | | | ... | -2 | -1 |

We can also specify a third argument.

Syntax: [**start** : **finish** : **step**]



specifies the step size to jump along the sequence


Defaults:

- **start**: beginning
- **finish**: end
- **step**: 1

Extended Slicing (Cont'd)

`myStr[::2]`  **'HloWrd'**

| | | | | | | | | | | | |
|------------|---|---|---|---|---|---|---|---|---|---|----|
| Characters | H | e | I | I | o | | W | o | r | I | d |
| Indices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |



Extended Slicing: Common Patterns

Copying a String

```
aString = "String to copy"
```

```
newStr = aString[:]
```

```
newStr = ''.join(aString)
```

Reversing a String

```
aString = "Madam I'm Adam"
```

```
revString = aString[::-1] → "maDA m' I madaM"
```

```
bString = aString[0:14:-1] → ""
```

Basic Operations

```
opStr = "Basic"
```

Length of a string: `len()`

e.g. `len(opStr)` → **5**

Concatenate strings: `+`

e.g. `opStr + "operations"` → **'Basic operations'**

Repeat String: `*`

e.g. `opStr * 3` → **'BasicBasicBasic'**

Two common systems for representing characters:

- **ASCII** (older)
- **Unicode** (modern, more characters)

Tables indicate mapping of characters to ASCII or Unicode.

ASCII vs. Unicode



ASCII

- Uses 8 bits to store a character
- $2^8 = 256$ different characters

Unicode

- An extension of ASCII
- Able to include more characters
- Uses 16 bits to store a character
- $2^{16} = 65,536$ characters

- The Unicode space is divided into 17 planes.
- Each plane contains 65,536 code points (16-bit).

*Total of:
1,114,112 characters,
96,000 used.*

ACSII Table (Partial)

| Dec | Hx | Oct | Html | Char | Dec | Hx | Oct | Html | Char | Dec | Hx | Oct | Html | Char | Dec | Hx | Oct | Html | Char | Dec | Hx | Oct | Html | Char |
|-----|----|-----|-------|-------|-----|----|-----|-------|------|-----|----|-----|-------|------|-----|----|-----|--------|------|-----|----|-----|--------|------|
| 32 | 20 | 040 | | Space | 52 | 34 | 064 | 4 | 4 | 72 | 48 | 110 | H | H | 92 | 5C | 134 | \ | \ | 112 | 70 | 160 | p | p |
| 33 | 21 | 041 | ! | ! | 53 | 35 | 065 | 5 | 5 | 73 | 49 | 111 | I | I | 93 | 5D | 135 |] |] | 113 | 71 | 161 | q | q |
| 34 | 22 | 042 | " | " | 54 | 36 | 066 | 6 | 6 | 74 | 4A | 112 | J | J | 94 | 5E | 136 | ^ | ^ | 114 | 72 | 162 | r | r |
| 35 | 23 | 043 | # | # | 55 | 37 | 067 | 7 | 7 | 75 | 4B | 113 | K | K | 95 | 5F | 137 | _ | _ | 115 | 73 | 163 | s | s |
| 36 | 24 | 044 | $ | \$ | 56 | 38 | 070 | 8 | 8 | 76 | 4C | 114 | L | L | 96 | 60 | 140 | ` | ` | 116 | 74 | 164 | t | t |
| 37 | 25 | 045 | % | % | 57 | 39 | 071 | 9 | 9 | 77 | 4D | 115 | M | M | 97 | 61 | 141 | a | a | 117 | 75 | 165 | u | u |
| 38 | 26 | 046 | & | & | 58 | 3A | 072 | : | : | 78 | 4E | 116 | N | N | 98 | 62 | 142 | b | b | 118 | 76 | 166 | v | v |
| 39 | 27 | 047 | ' | ' | 59 | 3B | 073 | ; | ; | 79 | 4F | 117 | O | O | 99 | 63 | 143 | c | c | 119 | 77 | 167 | w | w |
| 40 | 28 | 050 | (| (| 60 | 3C | 074 | < | < | 80 | 50 | 120 | P | P | 100 | 64 | 144 | d | d | 120 | 78 | 170 | x | x |
| 41 | 29 | 051 |) |) | 61 | 3D | 075 | = | = | 81 | 51 | 121 | Q | Q | 101 | 65 | 145 | e | e | 121 | 79 | 171 | y | y |
| 42 | 2A | 052 | * | * | 62 | 3E | 076 | > | > | 82 | 52 | 122 | R | R | 102 | 66 | 146 | f | f | 122 | 7A | 172 | z | z |
| 43 | 2B | 053 | + | + | 63 | 3F | 077 | ? | ? | 83 | 53 | 123 | S | S | 103 | 67 | 147 | g | g | 123 | 7B | 173 | { | { |
| 44 | 2C | 054 | , | , | 64 | 40 | 100 | @ | @ | 84 | 54 | 124 | T | T | 104 | 68 | 150 | h | h | 124 | 7C | 174 | | | |
| 45 | 2D | 055 | - | - | 65 | 41 | 101 | A | A | 85 | 55 | 125 | U | U | 105 | 69 | 151 | i | i | 125 | 7D | 175 | } | } |
| 46 | 2E | 056 | . | . | 66 | 42 | 102 | B | B | 86 | 56 | 126 | V | V | 106 | 6A | 152 | j | j | 126 | 7E | 176 | ~ | ~ |
| 47 | 2F | 057 | / | / | 67 | 43 | 103 | C | C | 87 | 57 | 127 | W | W | 107 | 6B | 153 | k | k | 127 | 7F | 177 | | DEL |
| 48 | 30 | 060 | 0 | 0 | 68 | 44 | 104 | D | D | 88 | 58 | 130 | X | X | 108 | 6C | 154 | l | l | | | | | |
| 49 | 31 | 061 | 1 | 1 | 69 | 45 | 105 | E | E | 89 | 59 | 131 | Y | Y | 109 | 6D | 155 | m | m | | | | | |
| 50 | 32 | 062 | 2 | 2 | 70 | 46 | 106 | F | F | 90 | 5A | 132 | Z | Z | 110 | 6E | 156 | n | n | | | | | |
| 51 | 33 | 063 | 3 | 3 | 71 | 47 | 107 | G | G | 91 | 5B | 133 | [| [| 111 | 6F | 157 | o | o | | | | | |

Source: www.lookupTables.com

Getting the Code

`ord()` takes a **character** as input and returns the Unicode of the character (for standard symbols this is the same as in ASCII).

e.g. `ord('a')` → **97**

`chr()` takes an ASCII/UTF-8 code and returns the corresponding character.

e.g. `chr(97)` → **'a'**

For encryption: `code = ord('a')`
`chr (code + 1)` → **'b'**

Comparing Two Characters

The code is used for comparison.

`'a' == 'a' → true`

`'a' < 'b' → true`

`'1' < '9' → true`

`'a' < 'B' → false!`



Is one string contained in another?

- Operator: **in**
- **a in b**: True if string **a** is contained in string **b**

```
myStr = "abcdefg"
```


```
'c' in myStr    → true
```

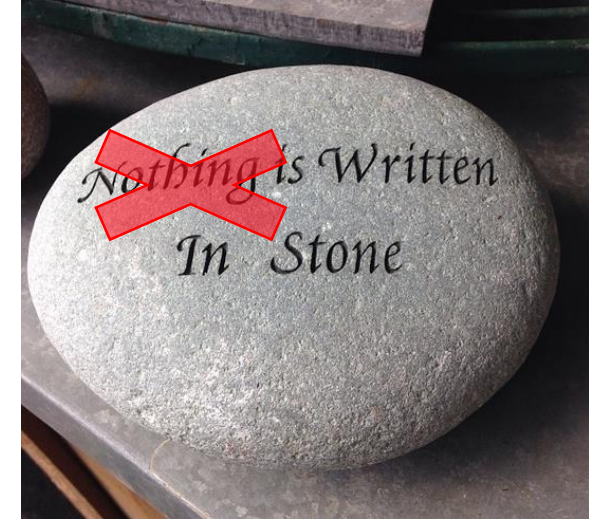
```
'cde' in myStr  → true
```

```
'cef' in myStr  → false
```

```
myStr in myStr → true
```

Strings are Immutable

- Strings are immutable, i.e., you cannot change one once you make it.
 - `aStr = 'spam'`
 - `aStr[1] = 1` →  Error
- However, you can use it to make another string (copy it, slice it, etc.).
 - `newStr = aStr[:1] + 'l' + aStr[2:]`
 - `newStr` → **'slam'**
 - `aStr` → **'spam'**



Functions and Methods for Strings

- A **function** is a **piece of code** that performs some operation.
 - The details are hidden (encapsulated), only it's interface exposed.
 - It is a way to arrange a program to make it **easier to understand**.
- A function has arguments as inputs and may return one output.

A String Function: `len()`



Recall Length of a string: `len()`

e.g. `len('test string')` → **11**
(not 10)

- Input: a string
- Output: an integer indicating the length of the string

- A method is a variation on a function.
 - It represents a program.
 - It has input arguments and output.
- Unlike a function, it is applied in the context of a particular object.
 - This is indicated by the **dot notation** invocation.
- Each string is itself an object.

String Method: `upper()`

`upper()` is a string method.

It will output a new string, which is the same as the string on which it was called, except all letters will now be in upper case.

```
myStr = "shouting!"  
myStr.upper() → 'SHOUTING!'
```

- Object: `myStr`
- Method: `upper()`
- Method call: `myStr.upper()`

Syntax: `object.method()`

- We say, **object** is calling the method **method**.

Different objects have different methods.



How do we find out all the methods available for strings?

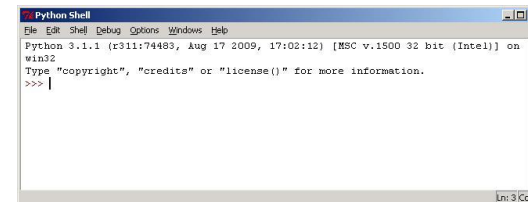
Use Reference

- **Python online**



<http://docs.python.org/release/3.2.3/library/stdtypes.html#string-methods>

- **Integrated Development Environment (IDE)** such as **IDLE**



String Method: `find()`

`find()` is another string method.

```
myStr = "Find in a string"  
myStr.find('d') → 3
```

→ 'd' is called an **argument** of the method.

- Input: a **single character**
- Output: the **index** of the character (first seen from left to right)
- If the character is not found, **-1** is returned

String Method: `join()`

`join()` is another string method.

```
str1 = "abcd"
str2 = "1234"
str2.join(str1) ➡ 'a1234b1234c1234d'
str1.join(str2) ➡ '1abcd2abcd3abcd4'
```

Syntax: `base.join(target)`

- Input: the **target string** to be joined
- Output: the new string where the **base** joins the **target**

Methods can be chained together.

Perform first operation, yielding an object.



Use the resulting object for the next method.

```
myStr = "Python Rules!"
```

```
myStr.upper()
```



'PYTHON RULES!'

```
myStr.upper().find('H')
```








3




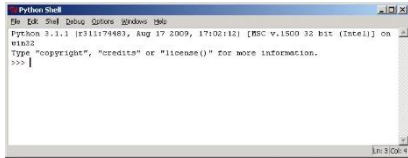
In this lesson, we have learnt:

- How to define a string in Python
- How to access characters in a Python string based on indices
- How to slice a Python string
- Common functions and methods for strings in Python

References for Images

| No. | Slide No. | Image | Reference |
|-----|-----------|---|--|
| 1 | 5 |  | String [Online Image]. Retrieved May 9, 2018 from https://www.flickr.com/photos/epublicist/8718123610 . |
| 2 | 6 |  | Search [Online Image]. Retrieved April 18, 2018 from https://pixabay.com/en/database-search-database-search-icon-2797375/ . |
| 3 | 7, 10 |  | By PKua - Own work, Public Domain, retrieved May 9, 2018 from https://commons.wikimedia.org/w/index.php?curid=3929297 . |
| 4 | 7, 13, 14 |  | Survey icon [Online Image]. Retrieved April 18, 2018 from https://pixabay.com/en/survey-icon-survey-icon-2316468/ . |
| 5 | 10 |  | Python Logo [Online Image]. Retrieved April 24, 2018 from https://pixabay.com/en/language-logo-python-2024210/ . |

References for Images

| No. | Slide No. | Image | Reference |
|-----|-----------|--|--|
| 6 | 24, 31 |  | Question problem [Online Image]. Retrieved April 24, 2018 from https://pixabay.com/en/question-problem-think-thinking-622164/ . |
| 7 | 25 |  | Written in stone [Online Image]. Retrieved May 9, 2018 from https://pixabay.com/en/nothing-is-written-in-stone-rock-527756/ . |
| 8 | 31 |  | World Wide Web [Online Image]. Retrieved May 9, 2018 from https://pixabay.com/en/world-wide-web-internet-computer-24958/ . |
| 9 | 31 |  | CC BY-SA 3.0, retrieved May 9, 2018 from https://commons.wikimedia.org/w/index.php?curid=11887635 . |