String

Presented By
M.Malarmathi
AP/IT

- A string is a **sequence of characters. You can access the characters one at a time with the** bracket operator

```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

- The second statement selects character number 1 from fruit and assigns it to letter
- The expression in brackets is called an **index. The index indicates which character in the** sequence you want (hence the name).

```
>>> letter = fruit[1.5]
```

- TypeError: string indices must be integers, not float

Len:

- len is a built-in function that returns the number of characters in a string

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

- To get the last character, you have to subtract 1 from length

```
>>> last = fruit[length-1]
>>> print last
a
```

String special operators

Ex: a=Hello b=Python

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

Traversal with a for loop

```
prefixes = 'JKLMNOPQ'
suffix = 'ack'
for letter in prefixes:
print letter + suffix
```

The output is:

Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack

String slices

- A segment of a string is called a **slice. Selecting a slice is similar to selecting a character**

```
>>> s = 'Monty Python'
>>> print s[0:5]
Monty
>>> print s[6:12]
Python
```

- If you omit the first index (before the colon), the slice starts at the beginning of the string.
- If you omit the second index, the slice goes to the end of the string:

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana‘
>>> fruit = 'banana'
>>> fruit[3:3]
‘’
```

- An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

Program Example

```
str = 'programiz
'print('str = ', str)
#first character
print('str[0] = ', str[0])
#last character
print('str[-1] = ', str[-1])
#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])
#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

OUTPUT

```
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
```

Strings are immutable

- The [] operator on the left side of an assignment, with the intention of changing a character in a string. For example:
- >>> greeting = 'Hello, world!'
- >>> greeting[0] = 'J'
- TypeError: 'str' object does not support item assignment
- The “object” in this case is the string and the “item” is the character you tried to assign.
- an **object is the same thing as a value, but we will refine that definition later. An item** is one of the values in a sequence.
- The reason for the error is that strings are **immutable, which means you can’t change an** existing string. The best you can do is create a new string that is a variation on the original:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print new_greeting
Jello, world!
```

- This example concatenates a new first letter onto a slice of greeting. It has no effect on the original string

String methods

- A **method** it takes arguments and returns a value
- For example, the method upper takes a string and returns a new string with all uppercase letters

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print new_word
BANANA
```

- The empty parentheses indicate that this method takes no argument.

A method call is called an **invocation**

- In this case upper on the word is invoked
- Example 1

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print index
1
```

Example 2

```
>>> word.find('na', 3)
4
```

- It can take as a second argument the index where it should start:

- Third argument the index where it should stop

```
>>> name = 'bulb'
>>> name.find('b', 1, 2)
-1
name = 'bob'
>>> name.find('b', 1, 2)
-1
```

- This search fails because b does not appear in the index range from 1 to 2

Built-in String Methods

- capitalize()
  Capitalizes first letter of string
- center(width, fillchar)
  Returns a space-padded string with the original string centered to a total of width columns.
- count(str, beg= 0,end=len(string))
  Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
- decode(encoding='UTF-8',errors='strict')
  Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.

- encode(encoding='UTF-8',errors='strict')
  Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
- endswith(suffix, beg=0, end=len(string))
  Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
- expandtabs(tabsize=8)
  Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
- find(str, beg=0 end=len(string))
  Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
- index(str, beg=0, end=len(string))
  Same as find(), but raises an exception if str not found.

- isalnum()
  Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
- isalpha()
  Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
- isdigit()
  Returns true if string contains only digits and false otherwise.
- islower()
  Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
- isnumeric()
  Returns true if a unicode string contains only numeric characters and false otherwise.

- isspace()
  Returns true if string contains only whitespace characters and false otherwise.
- istitle()
  Returns true if string is properly "titlecased" and false otherwise.
- isupper()
  Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
- join(seq)
  Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
- len(string)
  Returns the length of the string
- ljust(width[, fillchar])
  Returns a space-padded string with the original string left-justified to a total of width columns.
- lower()
  Converts all uppercase letters in string to lowercase.
- lstrip()
  Removes all leading whitespace in string.

String Modules

```
import string

text = "Monty Python's Flying Circus"

print "upper", "=>", string.upper(text)
print "lower", "=>", string.lower(text)
print "split", "=>", string.split(text)
print "join", "=>", string.join(string.split(text), "+")
print "replace", "=>", string.replace(text, "Python", "Java")
print "find", "=>", string.find(text, "Python"), string.find(text, "Java")
print "count", "=>", string.count(text, "n")
```

```
upper => MONTY PYTHON'S FLYING CIRCUS
lower => monty python's flying circus
split => ['Monty', "Python's", 'Flying', 'Circus']
join => Monty+Python's+Flying+Circus
replace => Monty Java's Flying Circus
find => 6 -1
count => 3
```

Example 2

```
text = "Monty Python's Flying Circus"

print "upper", "=>", text.upper()
print "lower", "=>", text.lower()
print "split", "=>", text.split()
print "join", "=>", "+".join(text.split())
print "replace", "=>", text.replace("Python", "Perl")
print "find", "=>", text.find("Python"), text.find("Perl")
print "count", "=>", text.count("n")
```

```
upper => MONTY PYTHON'S FLYING CIRCUS
lower => monty python's flying circus
split => ['Monty', "Python's", 'Flying', 'Circus']
join => Monty+Python's+Flying+Circus
replace => Monty Perl's Flying Circus
find => 6 -1
count => 3
```

Other Examples

```
>>> a = "this is a string"
>>> a = a.split(" ") # a is converted to a list of strings.
>>> print a
['this', 'is', 'a', 'string']
```

Joining a string is simple:

```
>>> a = "-".join(a)
>>> print a
this-is-a-string
```

Thank You