Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

`'hello'` is the same as `"hello"`.

You can display a string literal with the `print()` function:

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Or three single quotes:

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

```
a = "Hello, World!"
print(a[1])
```

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a `for` loop

Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

String Length

To get the length of a string, use the `len()` function.

Example

The `len()` function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

Check String

To check if a certain phrase or character is present in a string, we can use the keyword `in`.

Example

Check if "free" is present in the following text:

```
txt = "The best things in life are free!"
print("free" in txt)
```

Use it in an `if` statement:

Example

Print only if "free" is present:

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword `not in`.

Example

Check if "expensive" is NOT present in the following text:

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

Use it in an `if` statement:

Example

print only if "expensive" is NOT present:

```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

Note: The first character has index 0.

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 5 (not included):

```
b = "Hello, World!"
print(b[:5])
```

Slice To the End

By leaving out the *end* index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

```
b = "Hello, World!"
print(b[2:])
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "Hello, World!"
print(b[-5:-2])
```

Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

Upper Case

Example

The `upper()` method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

Lower Case

Example

The `lower()` method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The `strip()` method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Replace String

Example

The `replace()` method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Split String

The `split()` method returns a list where the text between the specified separator becomes the list items.

Example

The `split()` method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

Python - String Concatenation

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable `a` with variable `b` into variable `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Example

To add a space between them, add a `" "`:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

Python - Format - Strings

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

But we can combine strings and numbers by using the `format()` method!

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

Example

Use the `format()` method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers `{0}` to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Python - Escape Characters

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash `\` followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

To fix this problem, use the escape character `\"`:

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Escape Characters

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line

\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

Python - String Methods

String Methods

Python has a set of built-in methods that you can use on strings.

Note: All string methods returns new values. They do not change the original string.

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string

find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric

isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value

rfind()	Searches the string for a specified value and returns the last position o
rindex()	Searches the string for a specified value and returns the last position o
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa

title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning