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:

In [1]:
```
print("Hello")
print('Hello')
```

```
Hello
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:

In [2]:
```
a = "Hello"
print(a)
```

```
Hello
```

Multiline Strings

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

In [3]:
```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Or three single quotes:

In [4]:
```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Note: in the result, the line breaks are inserted at the same position as in the code.

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.

In [5]:
```
### Get the character at position 1
# (remember that the first character has the position 0):

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

```
e
```

Looping Through a String

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

In [15]:
```
# Loop through the letters in the word "banana":

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

```
b
a
n
a
n
a
```

String Length

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

In [16]:
```
### The len() function returns the length of a string:

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

```
13
```

Check String

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

In [17]:
```
# Check if "free" is present in the following text:

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

```
True
```

Use it in an if statement:

In [18]:
```
# Print only if "free" is present:

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

```
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.

In [19]:
```
# Check if "expensive" is NOT present in the following text:

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

```
True
```

Use it in an if statement:

In [21]:
```
# 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.")
```

```
No, 'expensive' is NOT present.
```

Python - Slicing Strings

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.

In [22]:
```
# Get the characters from position 2 to position 5 (not included):

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

```
llo
```

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:

In [23]:
```
# Get the characters from the start to position 5 (not included):

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

```
Hello
```

Slice To the End

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

In [24]:
```
# Get the characters from position 2, and all the way to the end:

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

```
llo, World!
```

Negative Indexing

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

Get the characters:

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

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

In [26]:
```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

Python - Modify Strings

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

Upper Case

In [27]:
```
# The upper() method returns the string in upper case:

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

```
HELLO, WORLD!
```

Lower Case

In [28]:
```
# The lower() method returns the string in lower case:

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

```
hello, world!
```

Remove Whitespace

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

In [29]:
```
# The strip() method removes any whitespace from the beginning or the end:

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

```
Hello, World!
```

Replace String

In [30]:
```
# The replace() method replaces a string with another string:

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

```
Jello, World!
```

Split String

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

In [33]:
```
# The split() method splits the string into substrings if it finds instances
# of the separator:

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

```
['Hello', ' World!']
```

Python - String Concatenation

String Concatenation

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

In [34]:
```
# Merge variable a with variable b into variable c:

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

```
HelloWorld
```

In [35]:
```
# To add a space between them, add a " ":

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

```
Hello World
```

Python - Format - Strings

String Format

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

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13400\982055754.py in <module>
      1 age = 36
----> 2 txt = "My name is John, I am " + age
      3 print(txt)

TypeError: can only concatenate str (not "int") to str
```

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:

In [37]:
```
# Use the format() method to insert numbers into strings:

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

```
My name is John, and I am 36
```

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

In [38]:
```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

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

In [39]:
```
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))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

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:

In [40]:
```
# 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."
```

```
  File "C:\Users\DELL\AppData\Local\Temp\ipykernel_13400\624176816.py", line
3
    txt = "We are the so-called "Vikings" from the north."
                                ^
SyntaxError: invalid syntax
```

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

In [41]:
```
txt = "We are the so-called \"Vikings\" from the north."
```

Other escape characters used in Python:

Code	Result	Try it
\'	Single Quote	Try it »
\\	Backslash	Try it »
\n	New Line	Try it »
\r	Carriage Return	Try it »
\t	Tab	Try it »
\b	Backspace	Try it »
\f	Form Feed	
\ooo	Octal value	Try it »
\xhh	Hex value	Try it »

In [42]:
```
#A backslash followed by three integers will result in a octal value:
txt = "\110\145\154\154\157"
print(txt)
```

```
Hello
```

In [44]:
```
#A backslash followed by an 'x' and a hex number represents a hex value:
txt = "\x48\x65\x6c\x6c\x6f"
print(txt)
```

```
Hello
```

In [47]:
```
txt = 'It\'s alright.'
print(txt)
```

```
It's alright.
```

In [48]:
```
txt = "This will insert one \\ (backslash)."
print(txt)
```

```
This will insert one \ (backslash).
```

In [49]:
```
txt = "Hello\nWorld!"
print(txt)
```

```
Hello
World!
```

In [50]:
```
txt = "Hello\rWorld!"
print(txt)
```

```
World!
```

In [51]:
```
txt = "Hello\tWorld!"
print(txt)
```

```
Hello	World!
```

In [1]:
```
print("Hello")
```

```
Hello
```

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

Upper case the first letter in this sentence:

In [2]:
```
txt = "hello, and welcome to my world."

x = txt.capitalize()

print (x)
```

```
Hello, and welcome to my world.
```

In [3]:
```
# The first character is converted to upper case, and the rest are converted to

txt = "python is FUN!"

x = txt.capitalize()

print (x)
```

```
Python is fun!
```

In [4]:
```
# See what happens if the first character is a number:

txt = "36 is my age."

x = txt.capitalize()

print (x)
```

```
36 is my age.
```

In [5]:
```
# Make the string lower case:

txt = "Hello, And Welcome To My World!"

x = txt.casefold()

print(x)
```

```
hello, and welcome to my world!
```

```
    Definition and Usage
    The casefold() method returns a string where all the characters are lo
    wer case.

    This method is similar to the lower() method, but the casefold() metho
    d is stronger, more aggressive, meaning that it will convert more char
    acters into lower case, and will find more matches when comparing two
    strings and both are converted using the casefold() method.
```

In [6]:
```
# Print the word "banana", taking up the space of 20 characters,\
# with "banana" in the middle:

txt = "banana"

x = txt.center(20)

print(x)
```

```
       banana
```

Definition and Usage

The center() method will center align the string, using a specified character (space is default) as the fill character.

In [7]:
```
# Using the letter "O" as the padding character:

txt = "banana"

x = txt.center(20, "O")

print(x)
```

```
OOOOOOObananaOOOOOOO
```

In [8]:
```
# Return the number of times the value "apple" appears in the string:

txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)
```

```
2
```

Definition and Usage

The count() method returns the number of times a specified value appears in the string.

In [9]:
```
# Search from position 10 to 24:

txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple", 10, 24)

print(x)
```

```
1
```

In [10]:
```
# UTF-8 encode the string:

txt = "My name is Ståle"

x = txt.encode()

print(x)
```

```
b'My name is St\xc3\xa5le'
```

Definition and Usage

The encode() method encodes the string, using the specified encoding. If no encoding is specified, UTF-8 will be used.

In [11]:
```
# These examples uses ascii encoding, and a character that cannot be encoded,
# showing the result with different errors:

txt = "My name is Ståle"

print(txt.encode(encoding="ascii",errors="backslashreplace"))
print(txt.encode(encoding="ascii",errors="ignore"))
print(txt.encode(encoding="ascii",errors="namereplace"))
print(txt.encode(encoding="ascii",errors="replace"))
print(txt.encode(encoding="ascii",errors="xmlcharrefreplace"))
```

```
b'My name is St\\xe5le'
b'My name is Stle'
b'My name is St\\N{LATIN SMALL LETTER A WITH RING ABOVE}le'
b'My name is St?le'
b'My name is St&#229;le'
```

Parameter	Description
encoding	Optional. A String specifying the encoding to use. Default is UTF-8
errors	Optional. A String specifying the error method. Legal values are:

`'backslashreplace'`	- uses a backslash instead of the character that could not be encoded
`'ignore'`	- ignores the characters that cannot be encoded
`'namereplace'`	- replaces the character with a text explaining the character
`'strict'`	- Default, raises an error on failure
`'replace'`	- replaces the character with a questionmark
`'xmlcharrefreplace'`	- replaces the character with an xml character

In [12]:
```
# Check if the string ends with a punctuation sign (.):

txt = "Hello, welcome to my world."

x = txt.endswith(".")

print(x)
```

```
True
```

Definition and Usage

The endswith() method returns True if the string ends with the specified value, otherwise False.

In [ ]:
```
Check if position 5 to 11 ends with the phrase "my world.":

txt = "Hello, welcome to my world."

x = txt.endswith("my world.", 5, 11)

print(x)
```

Python String expandtabs() Method

In [1]:
```
# Set the tab size to 2 whitespaces:

txt = "H\te\tl\tl\to"

x =  txt.expandtabs(2)

print(x)
```

```
H e l l o
```

In [2]:
```
txt = "H\te\tl\tl\to"

print(txt)
print(txt.expandtabs())
print(txt.expandtabs(2))
print(txt.expandtabs(4))
print(txt.expandtabs(10))
```

```
H       e       l       l       o
H       e       l       l       o
H e l l o
H   e   l   l   o
H         e         l         l         o
```

Python String find() Method

In [3]:
```
# Where in the text is the word "welcome"?:

txt = "Hello, welcome to my world."

x = txt.find("welcome")

print(x)
```

```
7
```

Definition and Usage

The find() method finds the first occurrence of the specified value.

The find() method returns -1 if the value is not found.

The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found. (See example below)

In [4]:
```
# Where in the text is the first occurrence of the letter "e"?:

txt = "Hello, welcome to my world."

x = txt.find("e")

print(x)
```

```
1
```

In [1]:
```
# Where in the text is the first occurrence of the letter "e"
# when you only search between position 5 and 10?:

txt = "Hello, welcome to my world."

x = txt.find("e", 5, 10)

print(x)
```

```
8
```

In [6]:
```
# If the value is not found, the find() method returns -1, but the index() meth

txt = "Hello, welcome to my world."

print(txt.find("q"))
print(txt.index("q"))
```

```
-1

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2504\670852428.py in <module>
      4
      5 print(txt.find("q"))
----> 6 print(txt.index("q"))

ValueError: substring not found
```

Python String format() Method

In [7]:
```
# Insert the price inside the placeholder, the price should be in fixed point,
# two-decimal format:

txt = "For only {price:.2f} dollars!"
print(txt.format(price = 49))
```

```
For only 49.00 dollars!
```

Definition and Usage

The format() method formats the specified value(s) and insert them inside the string's placeholder.

The placeholder is defined using curly brackets: {}. Read more about the placeholders in the Placeholder section below.

The format() method returns the formatted string.

The Placeholders

The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.

In [8]:
```
# Using different placeholder values:

txt1 = "My name is {fname}, I'm {age}".format(fname = "John", age = 36)
txt2 = "My name is {0}, I'm {1}".format("John",36)
txt3 = "My name is {}, I'm {}".format("John",36)
```

In [10]:
```
print(txt1)
print(txt2)
txt3
```

```
My name is John, I'm 36
My name is John, I'm 36
```

Out[10]:
```
"My name is John, I'm 36"
```

Formatting Types

Inside the placeholders you can add a formatting type to format the result:

`:<` Left aligns the result (within the available space)

`:>` Right aligns the result (within the available space)

`:^` Center aligns the result (within the available space)

`:=` Places the sign to the left most position

`:+` Use a plus sign to indicate if the result is positive or negative

`:-` Use a minus sign for negative values only

`: ` Use a space to insert an extra space before positive numbers (and a minus sign before negative numbers)

`:,` Use a comma as a thousand separator

`:_` Use a underscore as a thousand separator

`:b` Binary format

`:c` Converts the value into the corresponding unicode character

`:d` Decimal format

`:e` Scientific format, with a lower case e

`:E` Scientific format, with an upper case E

`:f` Fix point number format

`:F` Fix point number format, in uppercase format (show inf and nan as INF and NAN)

`:g` General format

`:G` General format (using a upper case E for scientific notations)

`:o` Octal format

`:x` Hex format, lower case

`:X` Hex format, upper case

`:n` Number format

`:%` Percentage format

In [11]:
```
#To demonstrate, we insert the number 8 to set the available space
# for the value to 8 characters.

#Use "<" to left-align the value:

txt = "We have {:<8} chickens."
print(txt.format(49))
```

```
We have 49       chickens.
```

In [12]:
```
#To demonstrate, we insert the number 8 to set the available space
# for the value to 8 characters.

#Use ">" to right-align the value:

txt = "We have {:>8} chickens."
print(txt.format(49))
```

```
We have       49 chickens.
```

In [13]:
```
#To demonstrate, we insert the number 8 to set the available space
# for the value to 8 characters.

#Use "^" to center-align the value:

txt = "We have {:^8} chickens."
print(txt.format(49))
```

```
We have    49    chickens.
```

In [14]:
```
#To demonstrate, we insert the number 8 to specify the available space
# for the value.

#Use "=" to place the plus/minus sign at the left most position:

txt = "The temperature is {:=8} degrees celsius."

print(txt.format(-5))
```

```
The temperature is -      5 degrees celsius.
```

In [15]:
```
#Use "+" to always indicate if the number is positive or negative:

txt = "The temperature is between {:+} and {:+} degrees celsius."

print(txt.format(-3, 7))
```

```
The temperature is between -3 and +7 degrees celsius.
```

In [16]:
```
#Use "-" to always indicate if the number is negative
#(positive numbers are displayed without any sign):

txt = "The temperature is between {:-} and {:-} degrees celsius."

print(txt.format(-3, 7))
```

```
The temperature is between -3 and 7 degrees celsius.
```

In [17]:
```
#Use " " (a space) to insert a space before positive numbers and
# a minus sign before negative numbers:

txt = "The temperature is between {: } and {: } degrees celsius."

print(txt.format(-3, 7))
```

```
The temperature is between -3 and  7 degrees celsius.
```

---

important

In [18]:
```
#Use "," to add a comma as a thousand separator:

txt = "The universe is {:,} years old."

print(txt.format(13800000000))
```

```
The universe is 13,800,000,000 years old.
```

In [20]:
```
#Use "_" to add a underscore character as a thousand separator:

txt = "The universe is {:_} years old."

print(txt.format(13800000000))
```

```
The universe is 13_800_000_000 years old.
```

In [21]:
```
#Use "b" to convert the number into binary format:

txt = "The binary version of {0} is {0:b}"

print(txt.format(5))
```

```
The binary version of 5 is 101
```

In [22]:
```
#Use "d" to convert a number, in this case a binary number,
# into decimal number format:

txt = "We have {:d} chickens."
print(txt.format(0b101))
```

```
We have 5 chickens.
```

In [23]:
```
#Use "e" to convert a number into scientific number format
# (with a lower-case e):

txt = "We have {:e} chickens."
print(txt.format(5))
```

```
We have 5.000000e+00 chickens.
```

In [24]:
```
#Use "E" to convert a number into scientific number format
# (with an upper-case E):

txt = "We have {:E} chickens."
print(txt.format(5))
```

```
We have 5.000000E+00 chickens.
```

important

In [25]:
```
#Use "f" to convert a number into a fixed point number,
#default with 6 decimals, but use a period followed by a number to specify
# the number of decimals:

txt = "The price is {:.2f} dollars."
print(txt.format(45))

#without the ".2" inside the placeholder,
# this number will be displayed like this:

txt = "The price is {:f} dollars."
print(txt.format(45))
```

```
The price is 45.00 dollars.
The price is 45.000000 dollars.
```

In [27]:
```
#Use "F" to convert a number into a fixed point number,
# but display inf and nan as INF and NAN:

x = float('inf')

txt = "The price is {:F} dollars."
print(txt.format(x))

#same example, but with a lower case f:

txt = "The price is {:f} dollars."
print(txt.format(x))
```

```
The price is INF dollars.
The price is inf dollars.
```

In [26]:
```
#Use "%" to convert the number into a percentage format:

txt = "You scored {:%}"
print(txt.format(0.25))

#Or, without any decimals:

txt = "You scored {:.0%}"
print(txt.format(0.25))
```

```
You scored 25.000000%
You scored 25%
```

In [ ]: