# Python

- String and its functionality
- String format
- Conditional statement if and else
- Nested if
- For loop basic introduction

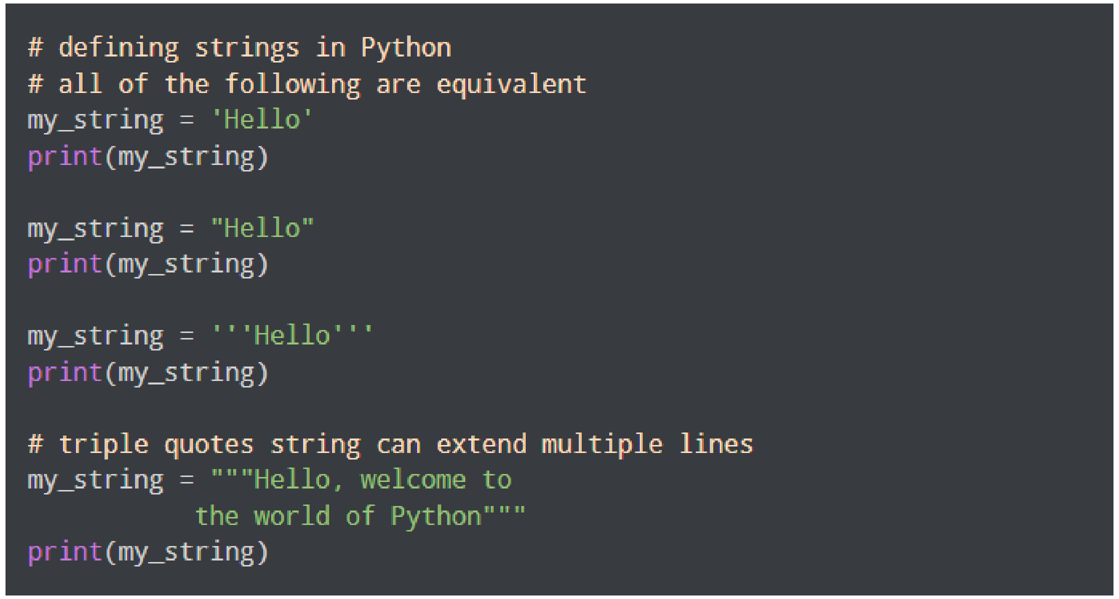
## String and its functionality

**What is String in Python?**

A string is a sequence of characters.

Strings can be created by enclosing characters inside a single quote or double-quotes.

Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.



```python
# defining strings in Python
# all of the following are equivalent
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
           the world of Python"""
print(my_string)
```

## How to access characters in a string?

- We can access individual characters using indexing and a range of characters using slicing.
- Index starts from 0. Trying to access a character out of index range will raise an IndexError.
- The index must be an integer. We can't use floats or other types, this will result into TypeError.

- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.
- We can access a range of items in a string by using the slicing operator :(colon).

```
#Accessing string characters in Python
str = ‘Python Programming'
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])
```

If we try to access an index out of the range or use numbers other than an integer, we will get errors.

```
# index must be in range
>>> my_string[15]
...
IndexError: string index out of range

# index must be an integer
>>> my_string[1.5]
...
TypeError: string indices must be integers
```

## Python String Operations

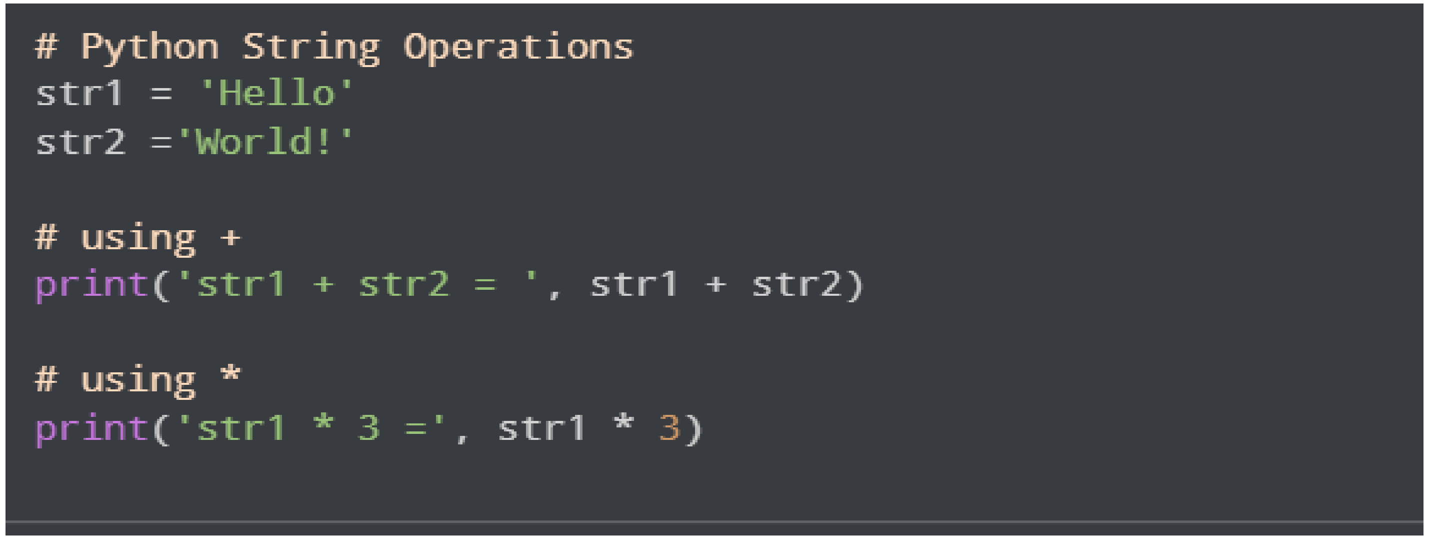
There are many operations that can be performed with strings which makes it one of the most used data types in Python.

## Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The **+** operator does this in Python. Simply writing two string literals together also concatenates them.

The ***** operator can be used to repeat the string for a given number of times.



```python
# Python String Operations
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

The `capitalize()` method returns a string where the first character is upper case, and the rest is lower case.

## Example

Upper case the first letter in this sentence:

```
txt = "hello, and welcome to my world."

x = txt.capitalize()

print (x)
```

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

## Syntax

```
string.center(length, character)
```

## Parameter Values

| **Parameter** | **Description** |
|---|---|
| *length* | Required. The length of the returned string |
| *character* | Optional. The character to fill the missing space on each side. Default is " " (space) |

## Example

Using the letter "O" as the padding character:

```
txt = "banana"

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

print(x)
```

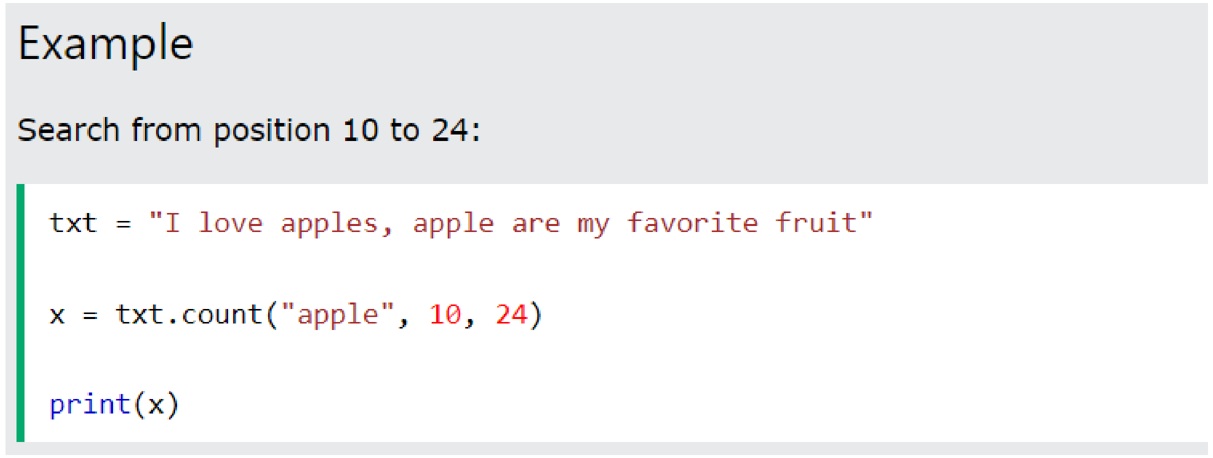
The `count()` method returns the number of times a specified value appears in the string.

## Syntax

```
string.count(value, start, end)
```

## Parameter Values

| Parameter | Description |
|---|---|
| *value* | Required. A String. The string to value to search for |
| *start* | Optional. An Integer. The position to start the search. Default is 0 |
| *end* | Optional. An Integer. The position to end the search. Default is the end of the string |



Example

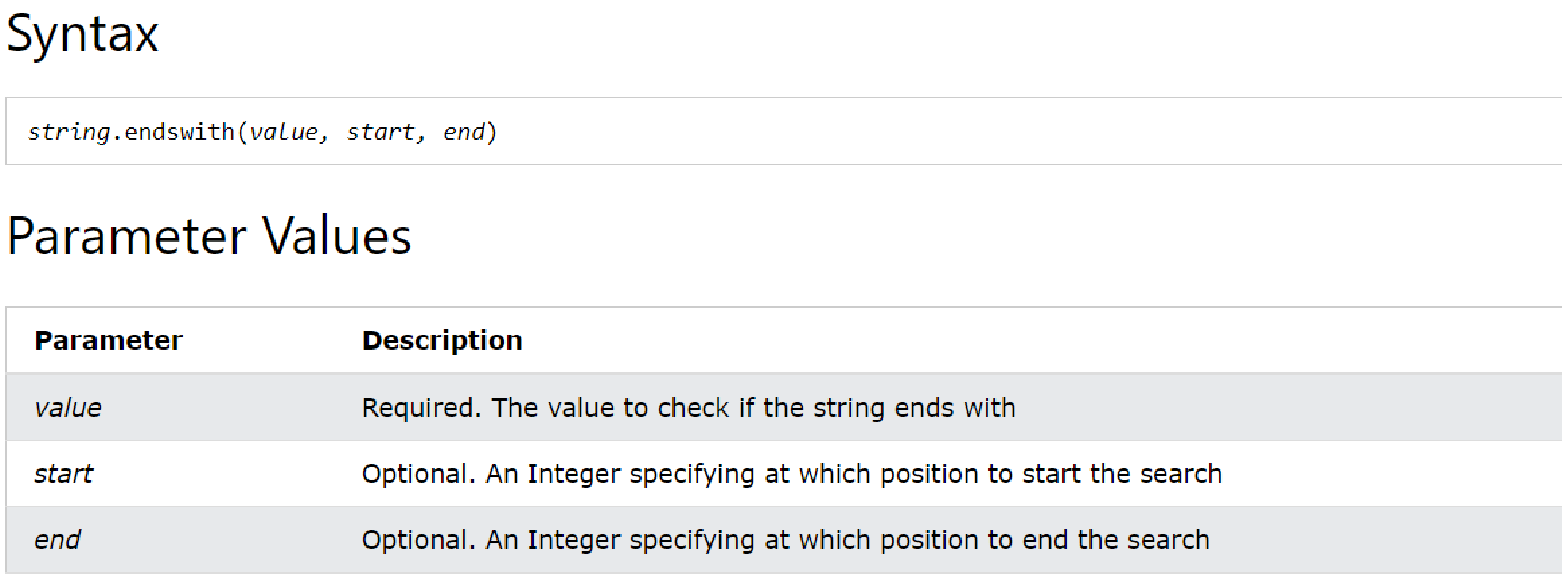
Search from position 10 to 24:

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

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

print(x)
```

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



## Syntax

```
string.endswith(value, start, end)
```

## Parameter Values

| **Parameter** | **Description** |
|---|---|
| *value* | Required. The value to check if the string ends with |
| *start* | Optional. An Integer specifying at which position to start the search |
| *end* | Optional. An Integer specifying at which position to end the search |

## Example

Check if the string ends with the phrase "my world.":

```
txt = "Hello, welcome to my world."

x = txt.endswith("my world.")

print(x)
```

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)

## Syntax

```
string.find(value, start, end)
```

## Parameter Values

| Parameter | Description |
|---|---|
| *value* | Required. The value to search for |
| *start* | Optional. Where to start the search. Default is 0 |
| *end* | Optional. Where to end the search. Default is to the end of the string |

### Example

Where in the text is the first occurrence of the letter "e"?:

```
txt = "Hello, welcome to my world."

x = txt.find("e")

print(x)
```

## Example

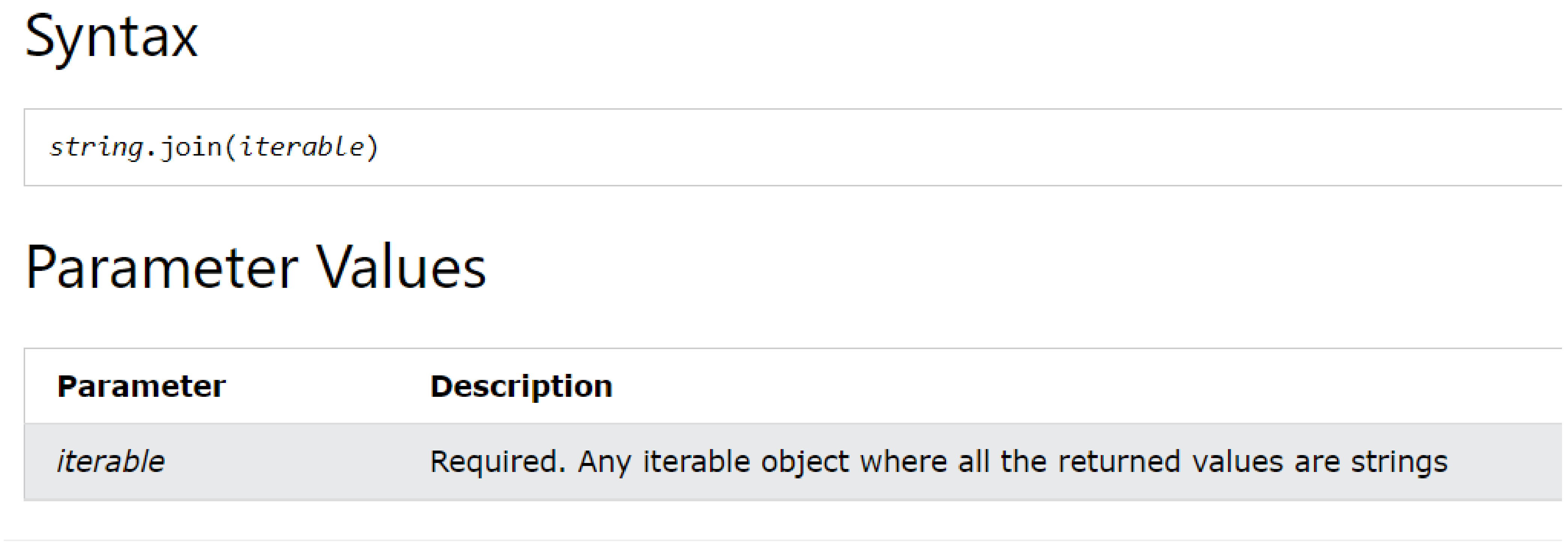
If the value is not found, the find() method returns -1, but the index() method will raise an exception:

```
txt = "Hello, welcome to my world."

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

| **isdecimal()** | **isdigit()** | **isnumeric()** |
|---|---|---|
| Example of string with decimal characters:<br>**"12345"**<br>**"12"**<br>**"98201"** | Example of string with digits:<br>**"12345"**<br>**"123$^3$"**<br>**"$^3$"** | Example of string with numerics:<br>**"12345"**<br>**"½¼"**<br>**"½"**<br>**"12345½"** |
| Returns ‘true’ if all characters of the string are decimal. | Returns ‘true’ if all characters of the string are digits. | Returns ‘true if all characters of the string are numeric. |

The `join()` method takes all items in an iterable and joins them into one string.
A string must be specified as the separator.

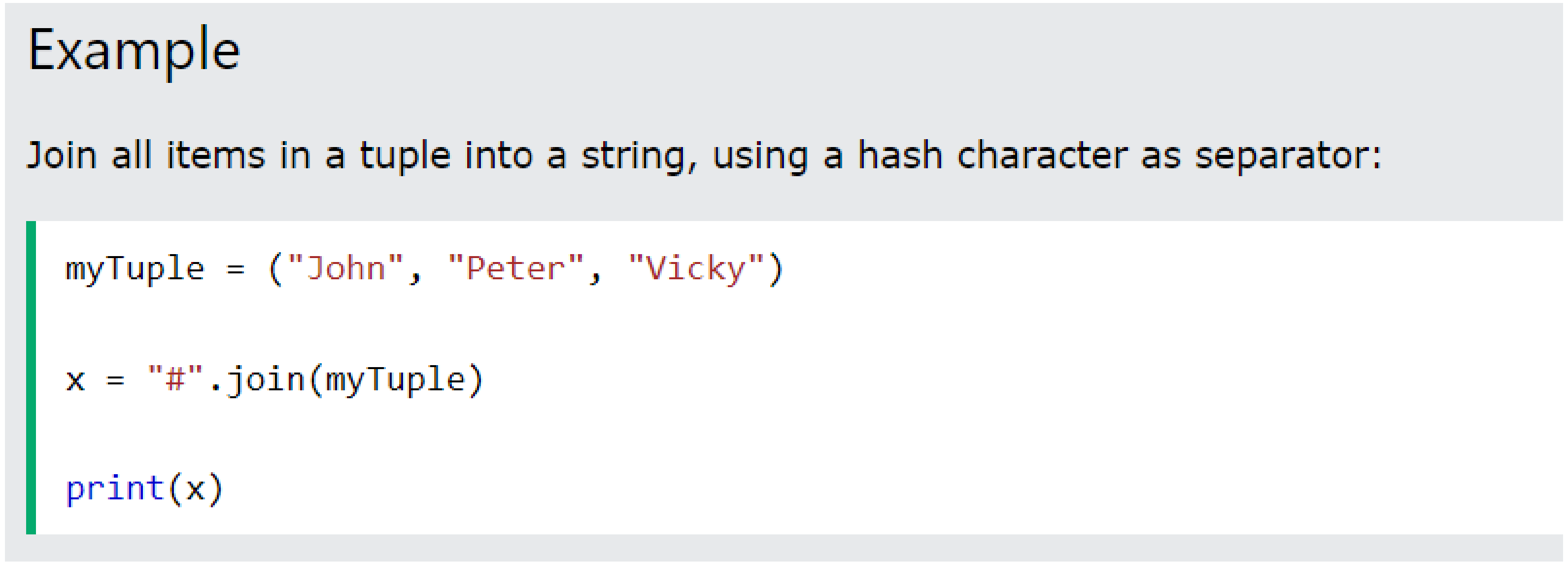


## Syntax

```
string.join(iterable)
```

## Parameter Values

| Parameter | Description |
|---|---|
| *iterable* | Required. Any iterable object where all the returned values are strings |



Example

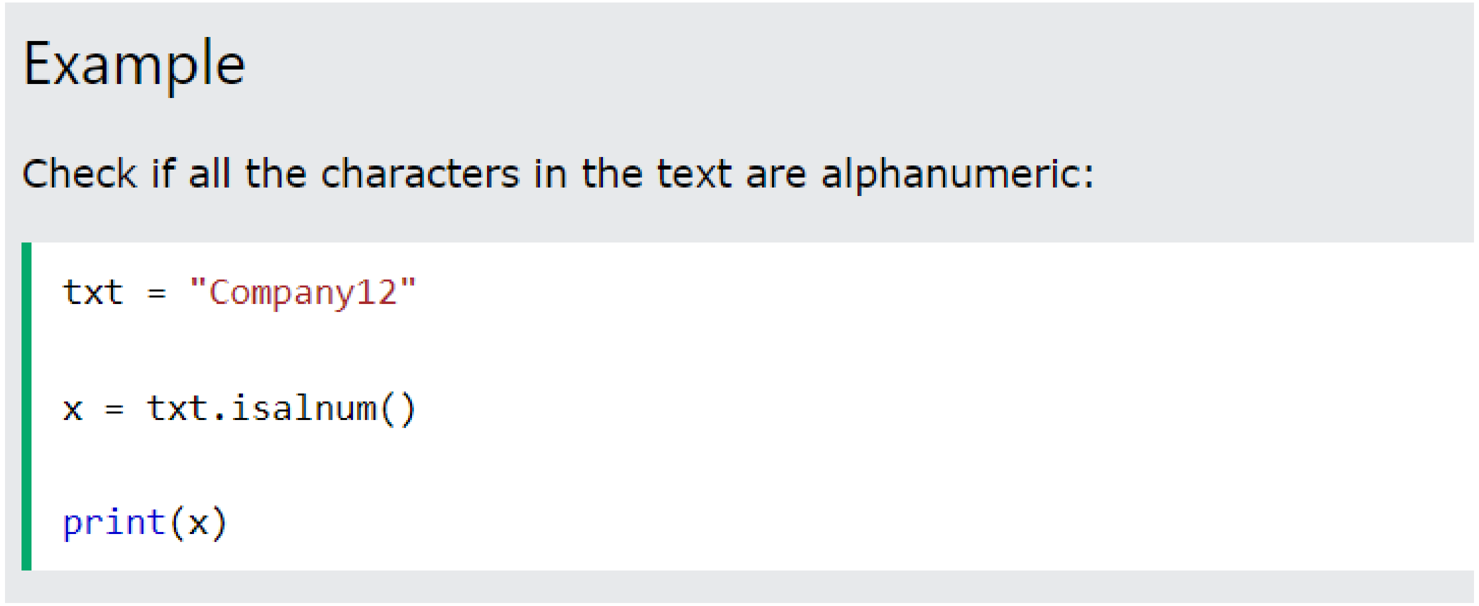
Join all items in a tuple into a string, using a hash character as separator:

```
myTuple = ("John", "Peter", "Vicky")

x = "#".join(myTuple)

print(x)
```

## Isalnum()



Example

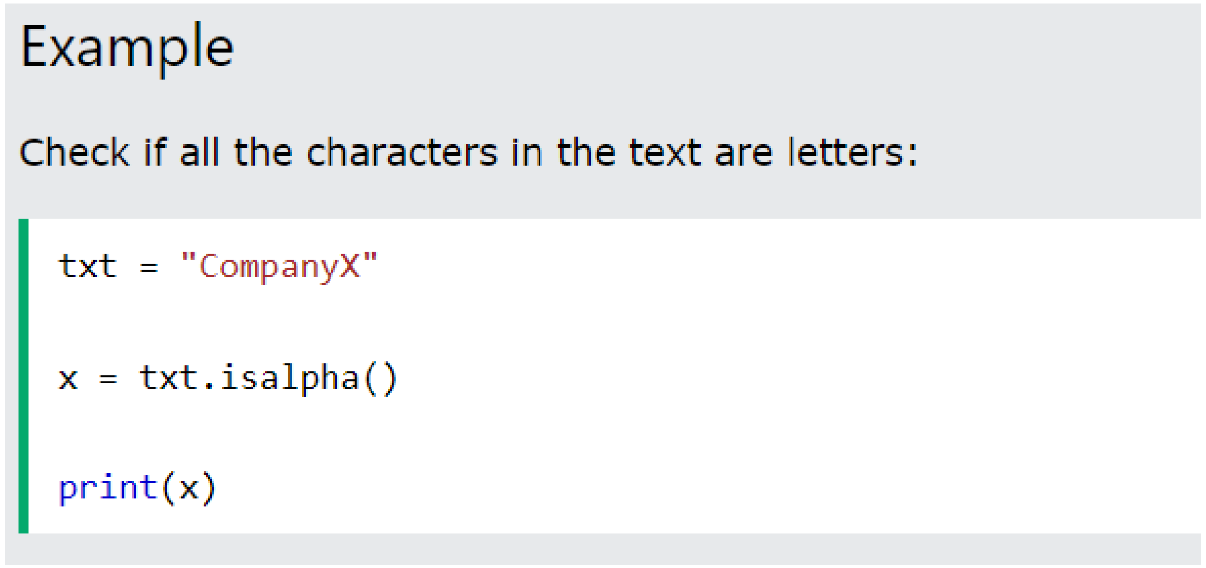
Check if all the characters in the text are alphanumeric:

```
txt = "Company12"

x = txt.isalnum()

print(x)
```

## isalpha()



Example

Check if all the characters in the text are letters:

```
txt = "CompanyX"

x = txt.isalpha()

print(x)
```

## Iterating Through a string

We can iterate through a string using a for loop. Here is an example to count the number of 'l's in a string.

```
# Iterating through a string
count = 0
for letter in 'Hello World':
    if(letter == 'l'):
        count += 1
print(count,'letters found')
```

## String Membership Test

We can test if a substring exists within a string or not, using the keyword `in`.

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

## Practice Questions if and else

A company decided to give bonus of 5% to employee if his/her year of service is more than 5 years.
Ask user for their salary and year of service and print the net bonus amount.

Take two int values from user and print greatest among them.

A shop will give discount of 10% if the cost of purchased quantity is more than 1000.
Ask user for quantity
Suppose, one unit will cost 100.
Judge and print total cost for user.

**Write a program to calculate the electricity bill (accept number of unit from user) according to the following criteria :**
**Unit Price First 100 units no charge Next 100 units Rs 5 per unit After 200 units Rs 10 per unit**
**(For example if input unit is 350 than total bill amount is Rs2000)**