תכנות מונחה עצמים
Python

נכתב ע"י: אוהד שירזי

Interpreted language

An interpreted language is a type of programming language for which most of its implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions. The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines, and then into another language (often machine code).

Advantages of interpreted languages are:

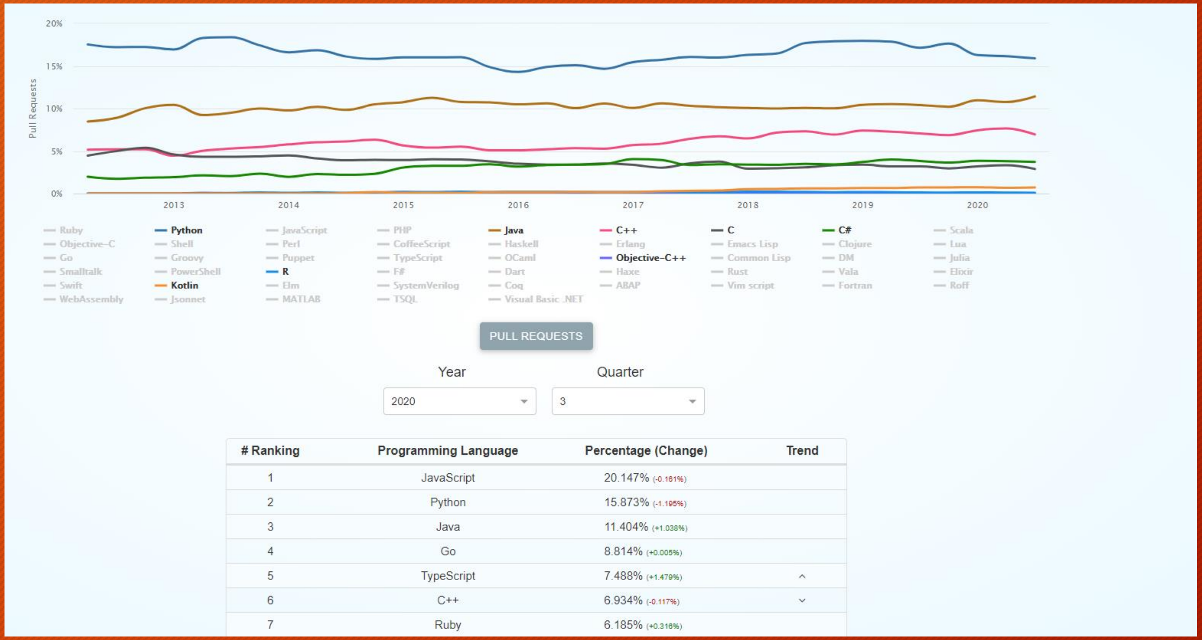
- Interpreting a language gives implementations some additional flexibility over compiled implementations. Features that are often easier to implement in interpreters than in compilers include:
- platform independence (Java's byte code, for example)
- reflection and reflective use of the evaluator (e.g. a first-order eval function)
- dynamic typing
- smaller executable program size (since implementations have flexibility to choose the instruction code)
- dynamic scoping
- Furthermore, source code can be read and copied, giving users more freedom.

Disadvantages of interpreted languages are:

- Without static type-checking, which is usually performed by a compiler, programs can be less reliable,[citation needed] because type checking eliminates a class of programming errors (though type-checking of the code can be done by using additional stand-alone tools. See TypeScript for instance)
- Interpreters can be susceptible to Code injection attacks.
- Slower execution compared to direct native machine code execution on the host CPU. A technique used to improve performance is just-in-time compilation which converts frequently executed sequences of interpreted instruction to host machine code. JIT is most often combined with compilation to byte-code as in Java.
- Source code can be read and copied (e.g. JavaScript in web pages), or more easily reverse engineered through reflection in applications where intellectual property has a commercial advantage. In some cases, obfuscation is used as a partial defense against this.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python has a simple syntax like the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.



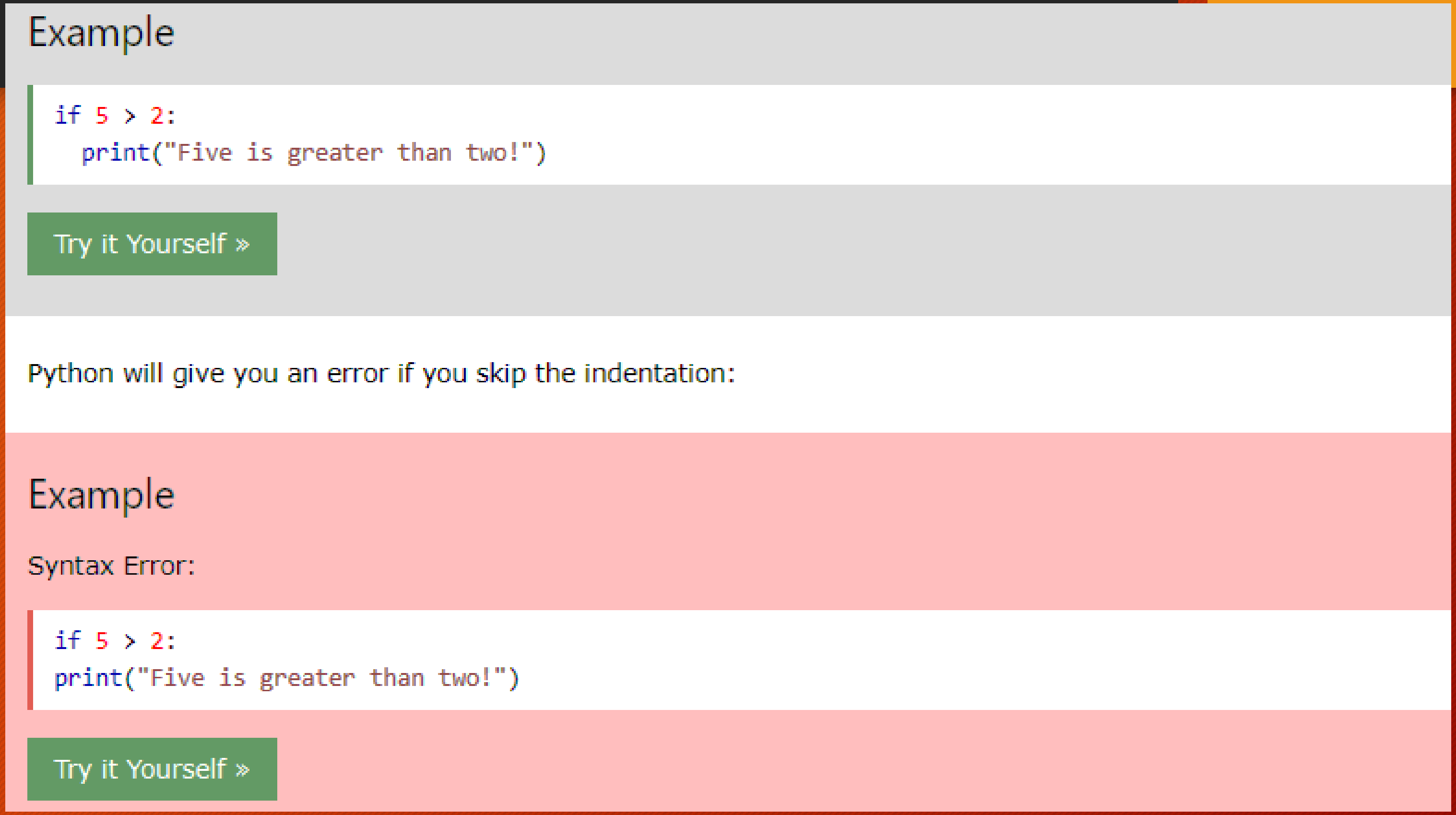
Year: 2020

Quarter: 3

# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	20.147% (-0.161%)	
2	Python	15.873% (-1.195%)	
3	Java	11.404% (+1.038%)	
4	Go	8.814% (+0.005%)	
5	TypeScript	7.488% (+1.479%)	^
6	C++	6.934% (-0.117%)	˅
7	Ruby	6.185% (+0.316%)	

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.



Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

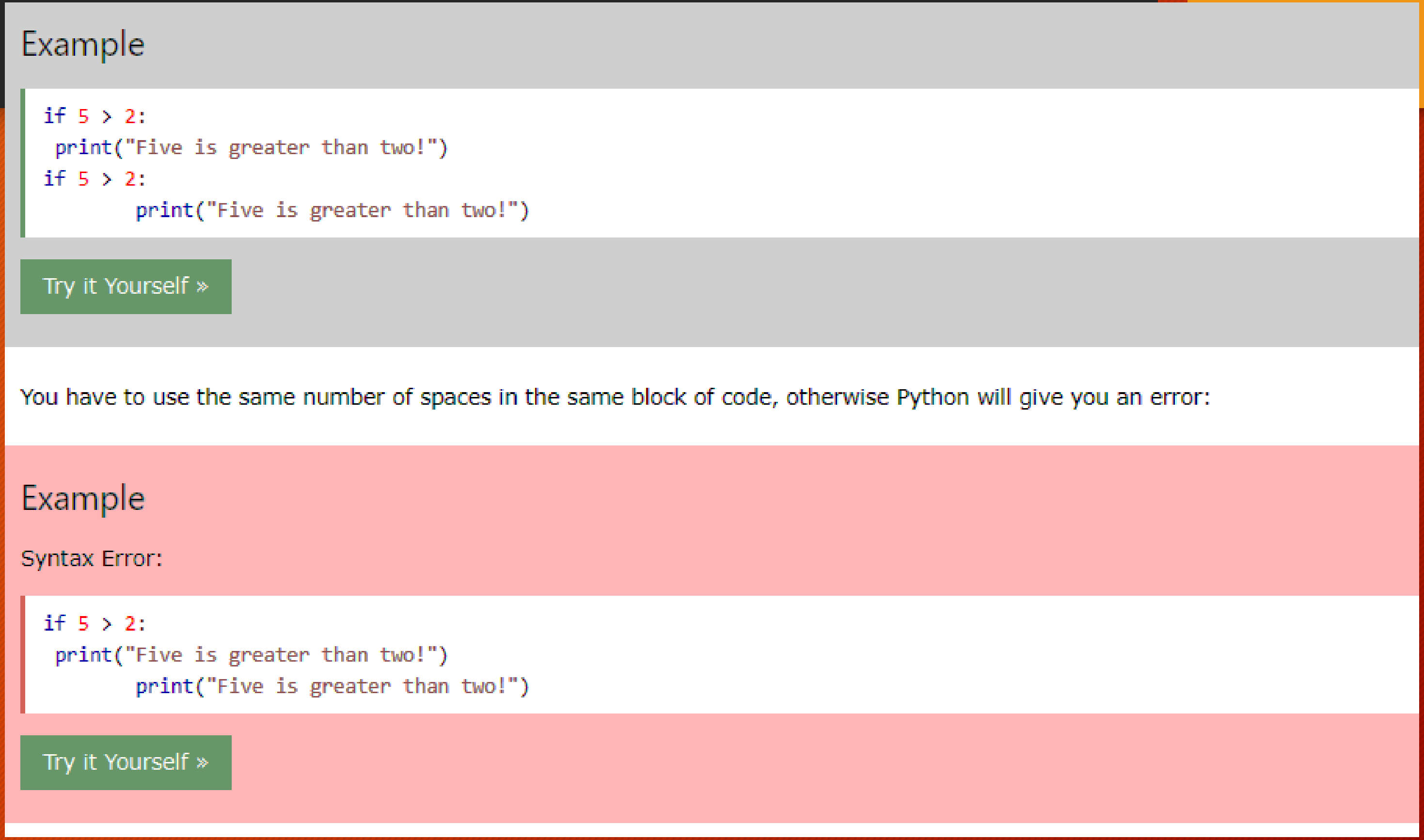
```
if 5 > 2:
print("Five is greater than two!")
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.



Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

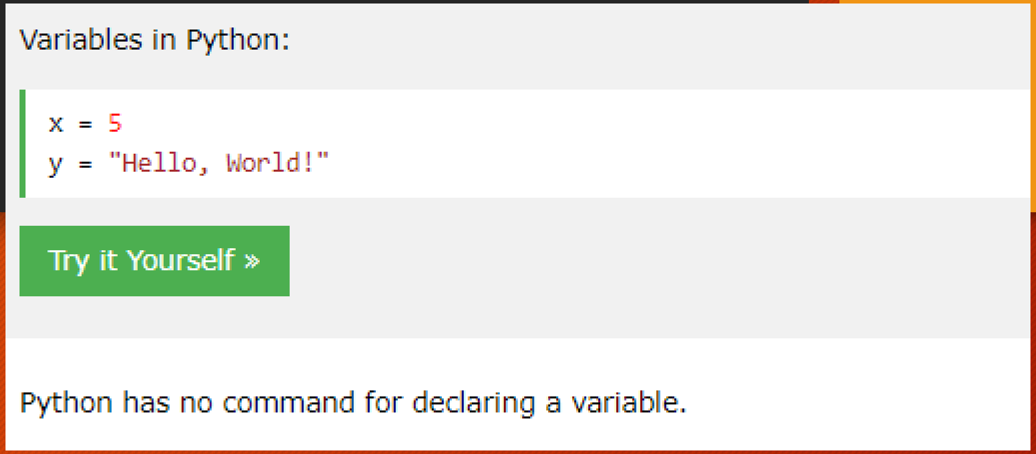
Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

Python syntax basics

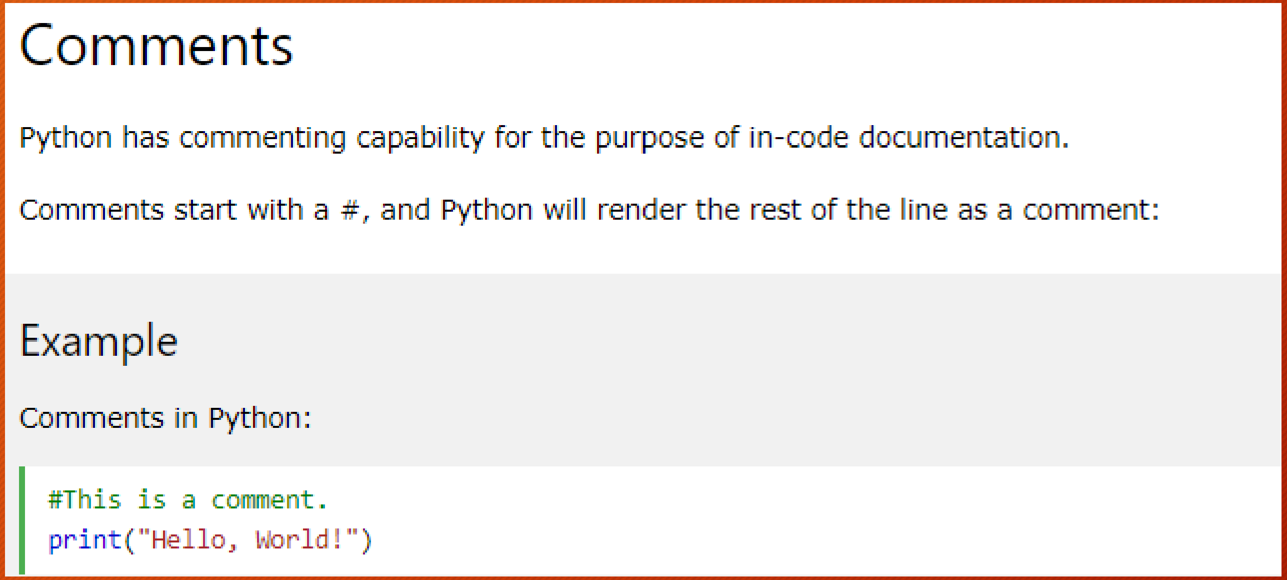
In Python, variables are created when you assign a value to it



Variables in Python:

```
x = 5
y = "Hello, World!"
```

Python has no command for declaring a variable.



Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

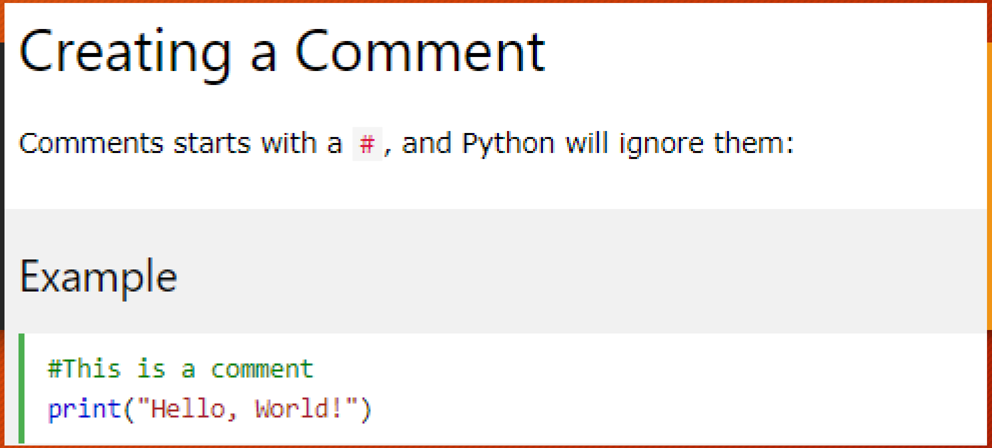
Example

Comments in Python:

```
#This is a comment.
print("Hello, World!")
```

Python Comments

Comments can be used to explain Python code.
Comments can be used to make the code more readable.
Comments can be used to prevent execution when testing code.

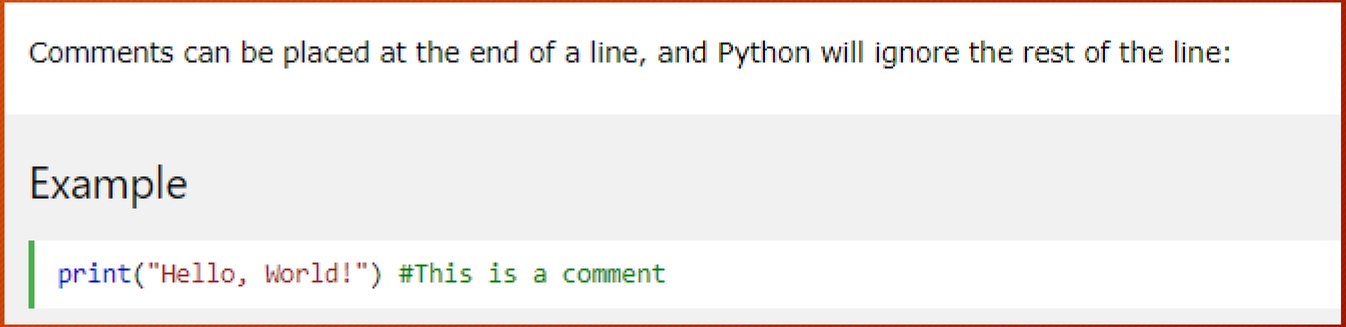


Creating a Comment

Comments starts with a `#`, and Python will ignore them:

Example

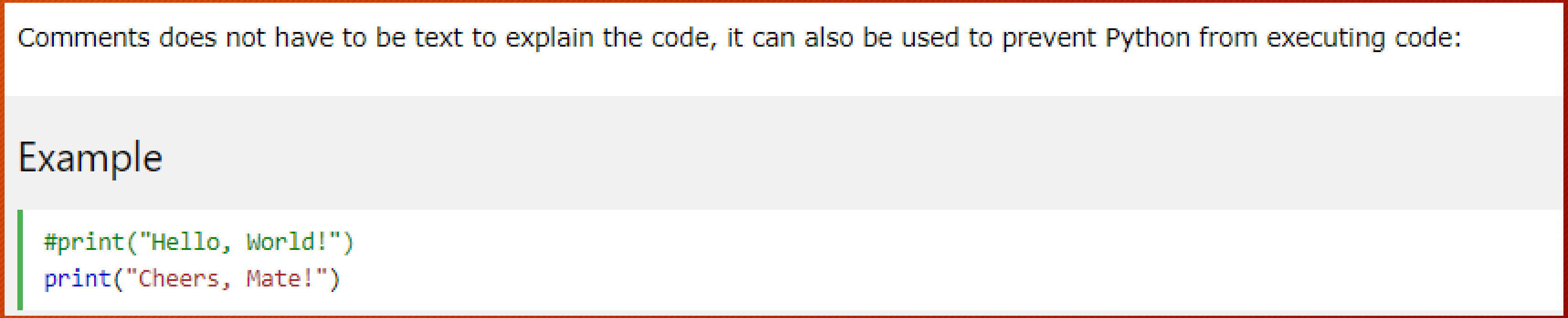
```
#This is a comment
print("Hello, World!")
```



Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

```
print("Hello, World!") #This is a comment
```



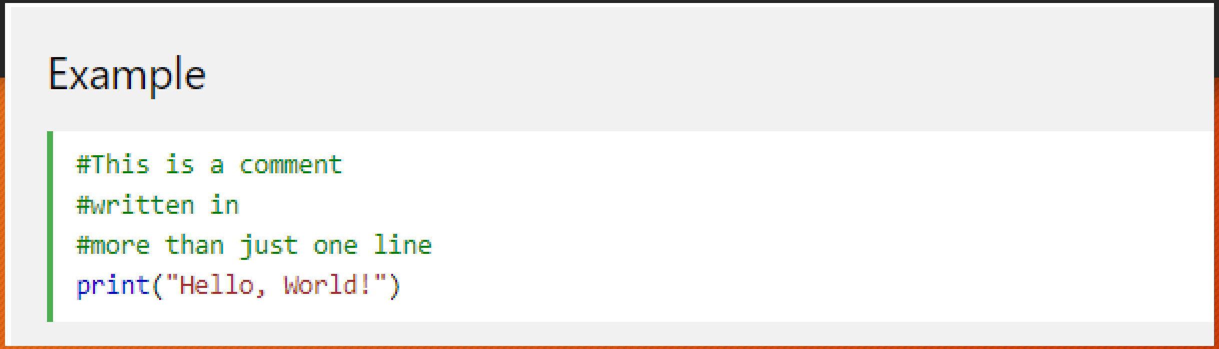
Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code:

Example

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multi Line Comments

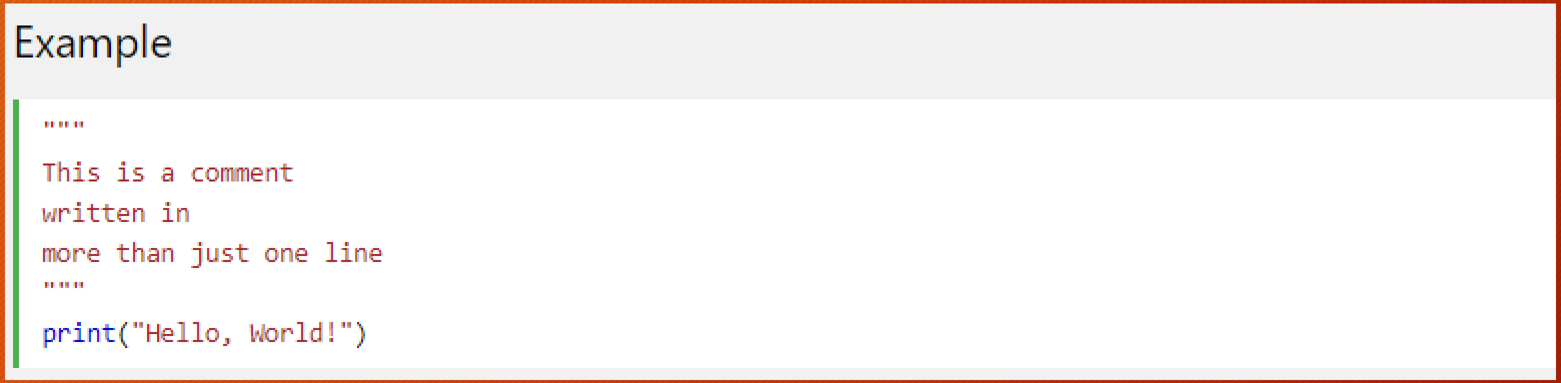
First way for a Multi Line Comments



Example

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Second way for a Multi Line Comments



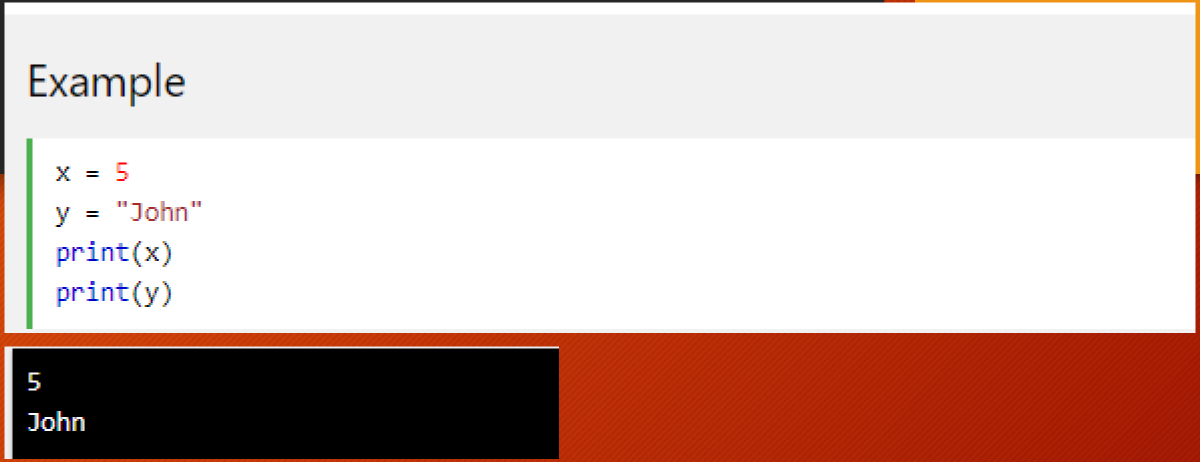
Example

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Python Variables

Creating Variables
Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

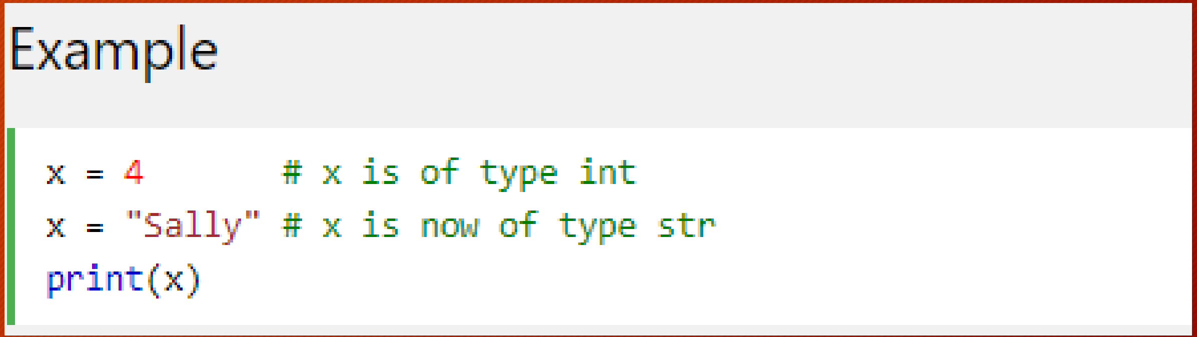


Example

```python
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

Variables do not need to be declared with any particular type, and can even change type after they have been set.

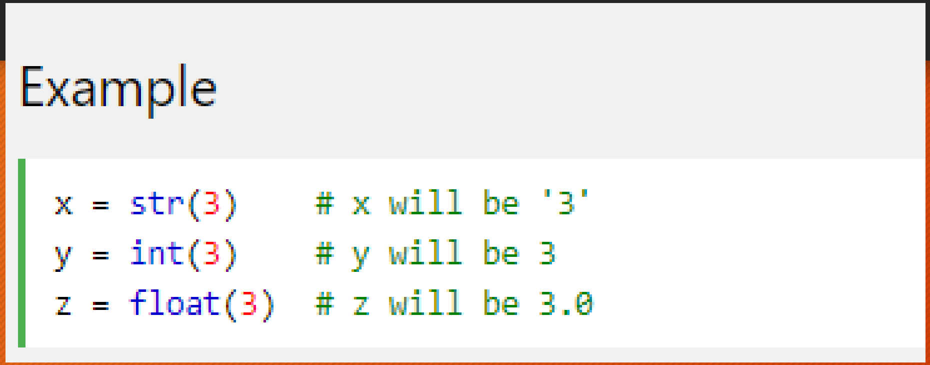


Example

```python
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

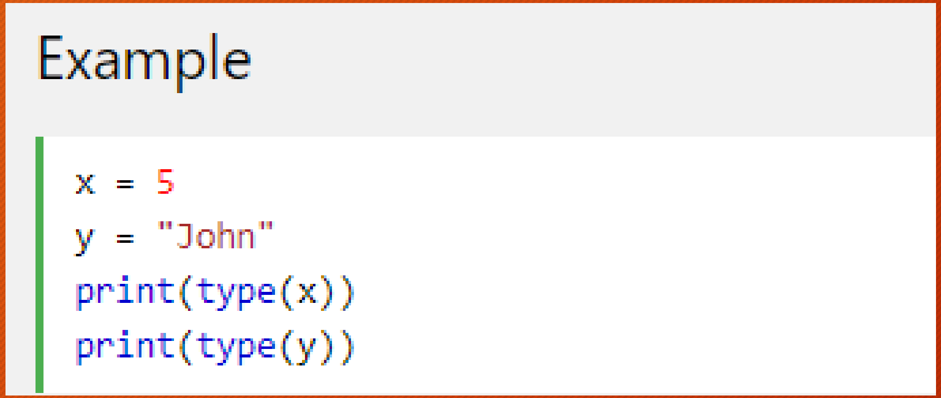
Python Variables

If you want to specify the data type of a variable, this can be done with casting.


Example

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

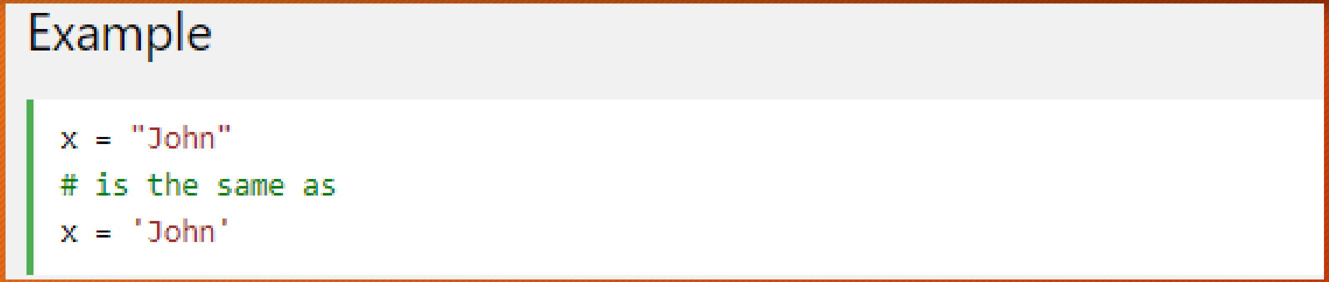
Get the Type
You can get the data type of a variable with the type() function.


Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

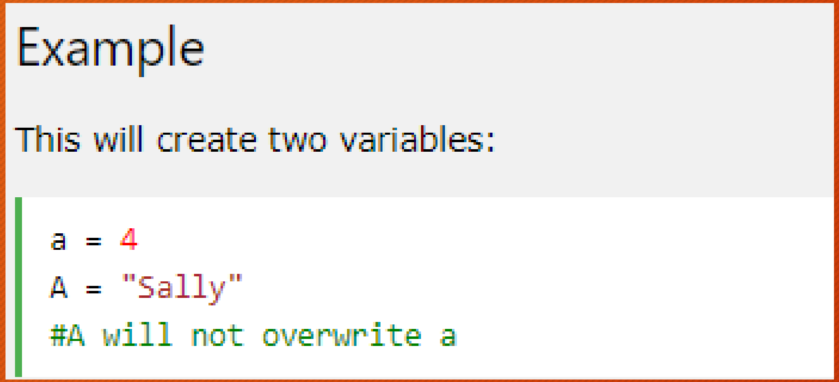
Python Variables

Single or Double Quotes? (Same as java)
String variables can be declared either by using single or double quotes:


Example

```
x = "John"
# is the same as
x = 'John'
```

Case-Sensitive (Same as java)
Variable names are case-sensitive.


Example

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

Python - Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

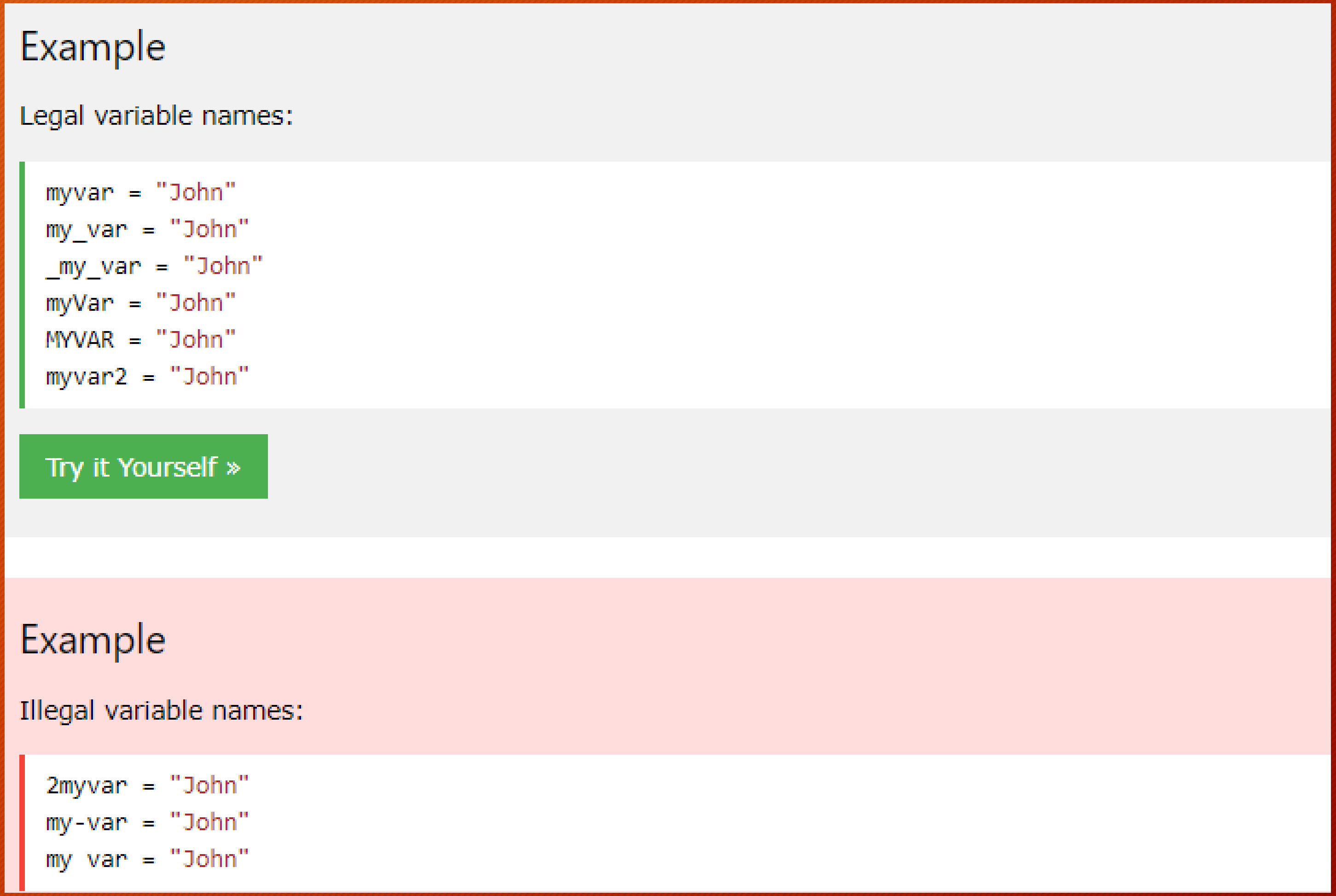
Rules for Python variables:

A variable name **must** start with a letter or the underscore character

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (age, Age and AGE are three different variables)



Example

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

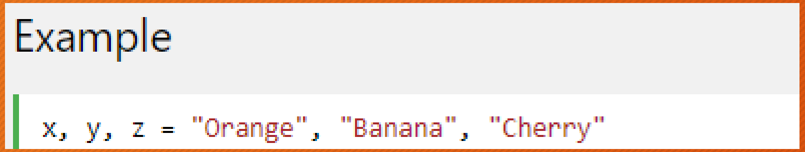
Example

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

Python Variables - Assign Multiple Values

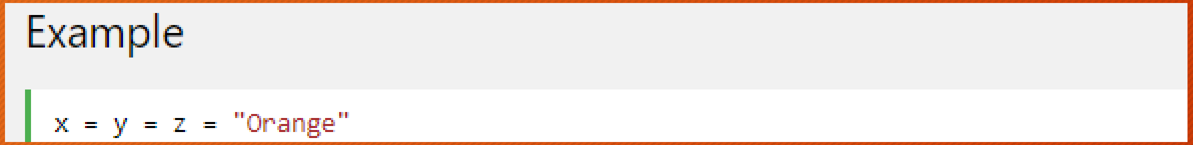
Many Values to Multiple Variables
Python allows you to assign values to multiple variables in one line:



Example

```
x, y, z = "Orange", "Banana", "Cherry"
```

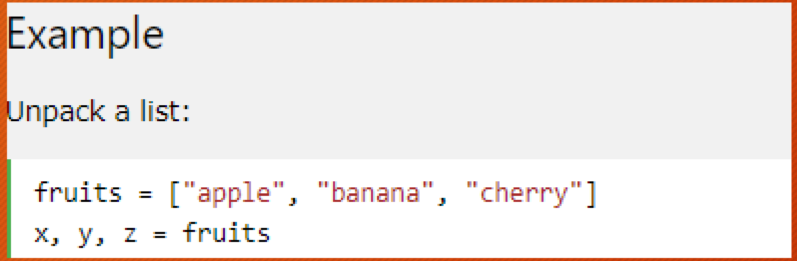
One Value to Multiple Variables
And you can assign the same value to multiple variables in one line:



Example

```
x = y = z = "Orange"
```

Unpack a Collection
If you have a collection of values in a list, tuple etc. Python allows you extract the values into variables. This is called unpacking.



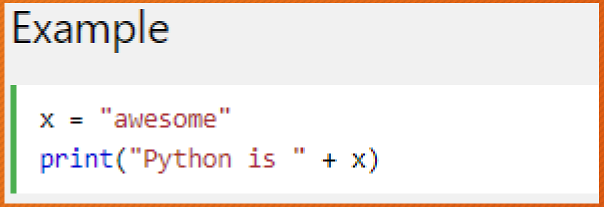
Example

Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
```

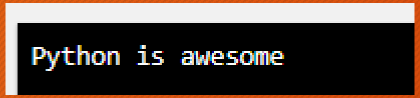
Output Variables

The Python print statement is often used to output variables.
To combine both text and a variable, Python uses the + character:



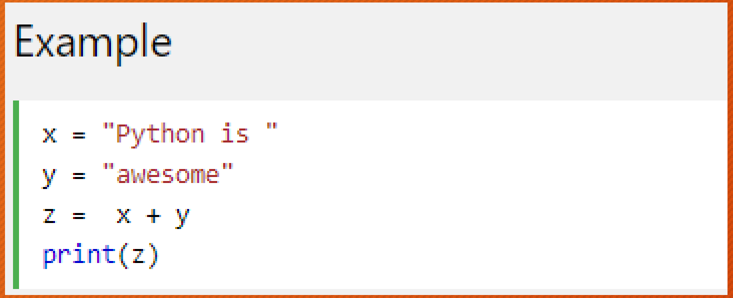
Example

```
x = "awesome"
print("Python is " + x)
```



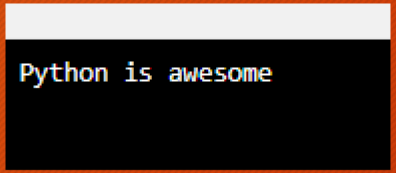
```
Python is awesome
```

You can also use the + character to add a variable to another variable:



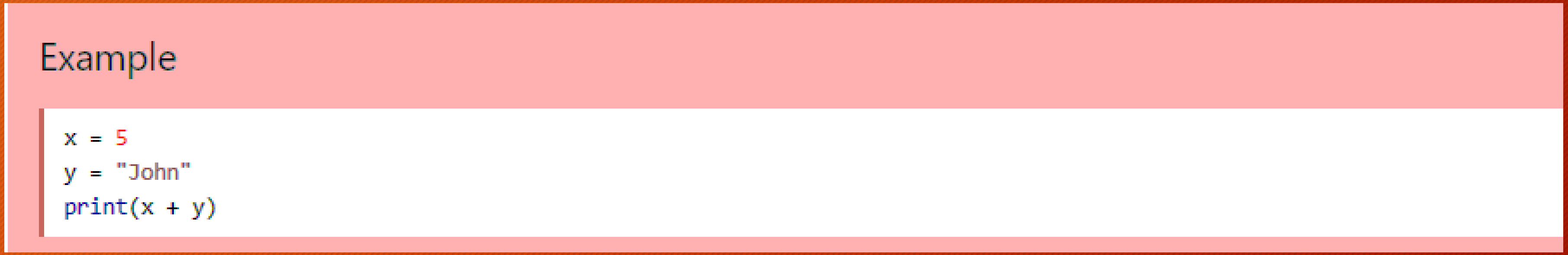
Example

```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```



```
Python is awesome
```

If you try to combine a string and a number, Python will give you an error:



Example

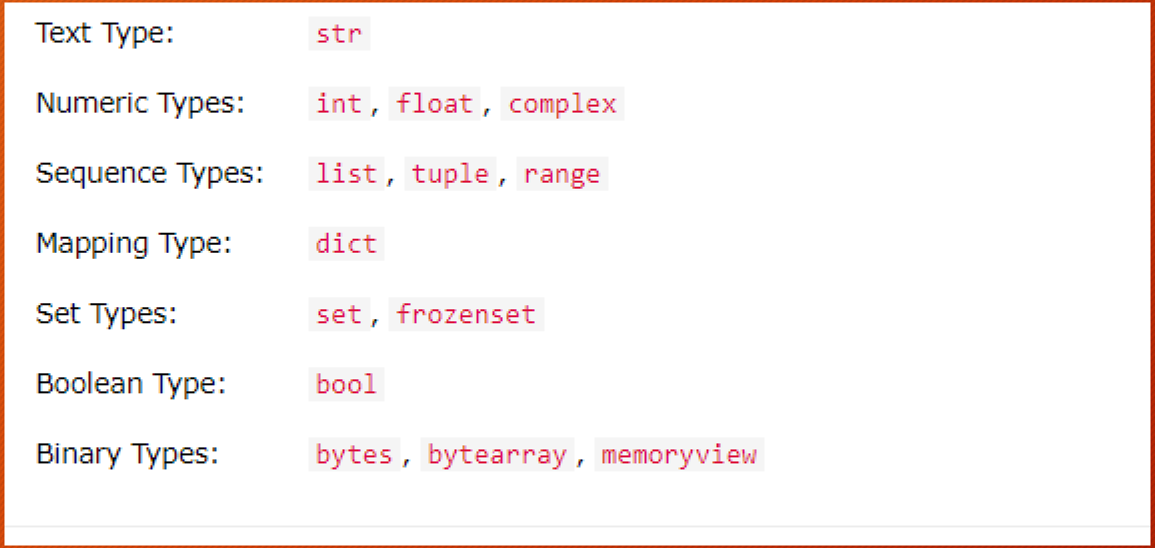
```
x = 5
y = "John"
print(x + y)
```

Python Data Types

Built-in Data Types
In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

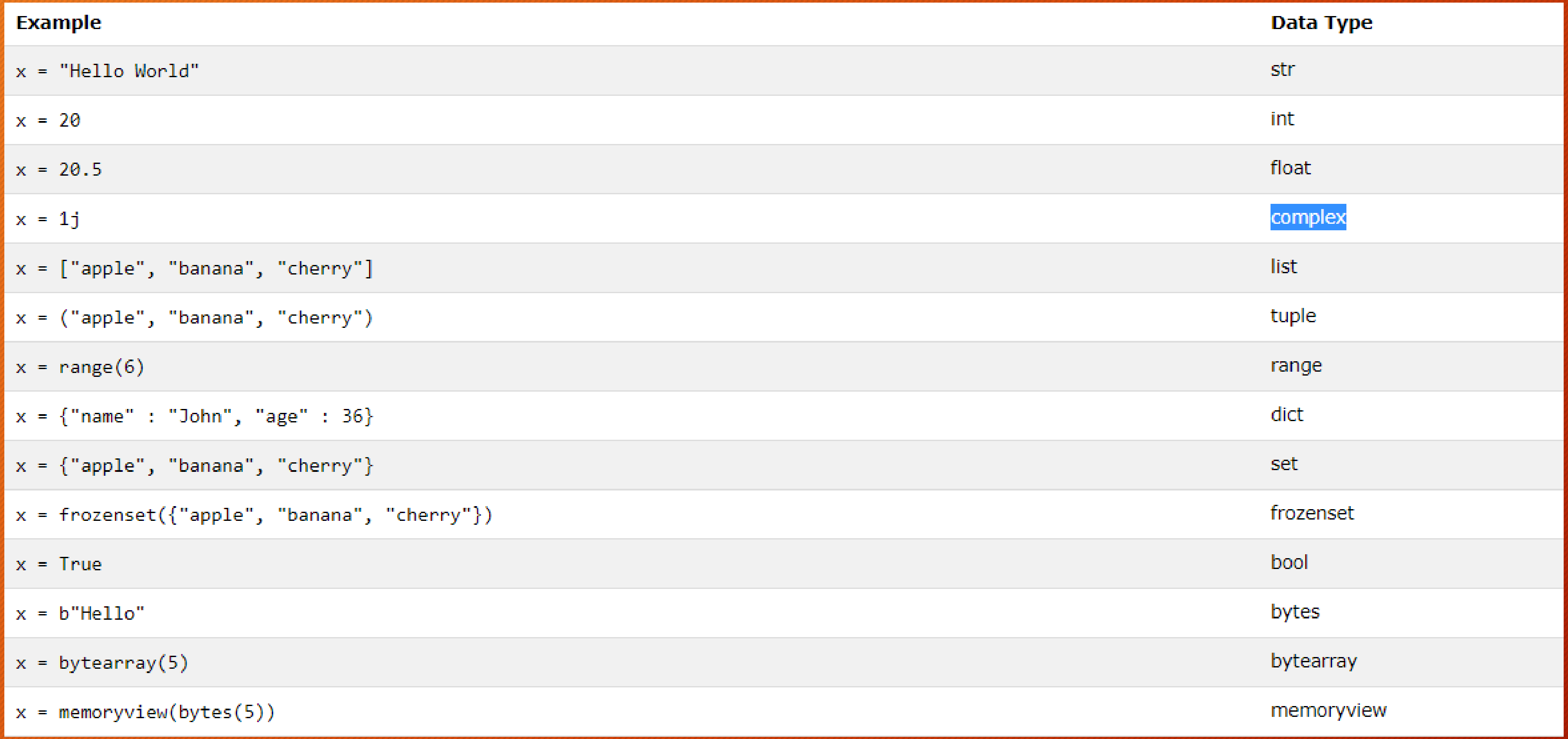
Python has the following data types built-in by default, in these categories:



Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

Python Data Types Cont.

In Python, the data type is set when you assign a value to a variable:

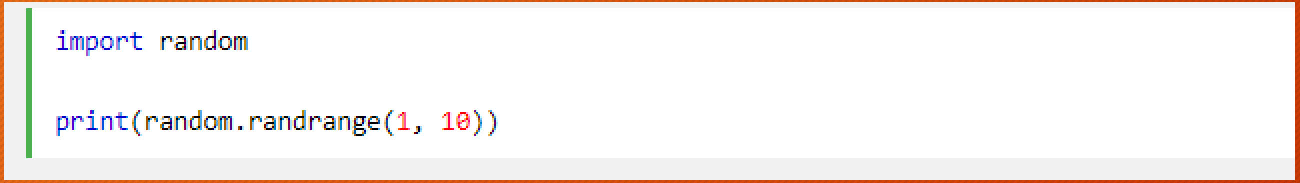


Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Random Number

Random Number
Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

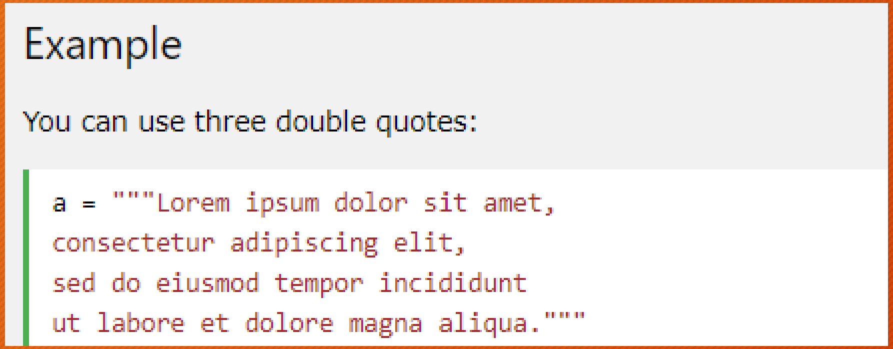
Example
Import the random module, and display a random number between 1 and 9:



```
import random

print(random.randrange(1, 10))
```

Multiline Strings

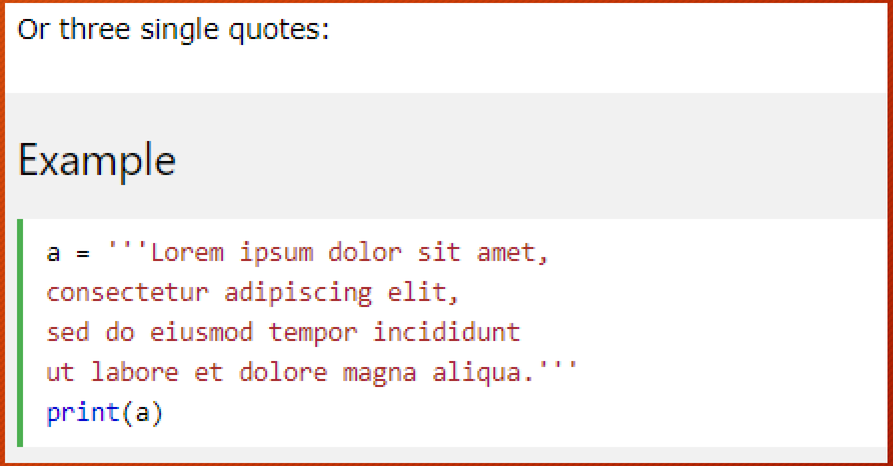
You can assign a multiline string to a variable by using three quotes:



Example

You can use three double quotes:

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



Or three single quotes:

Example

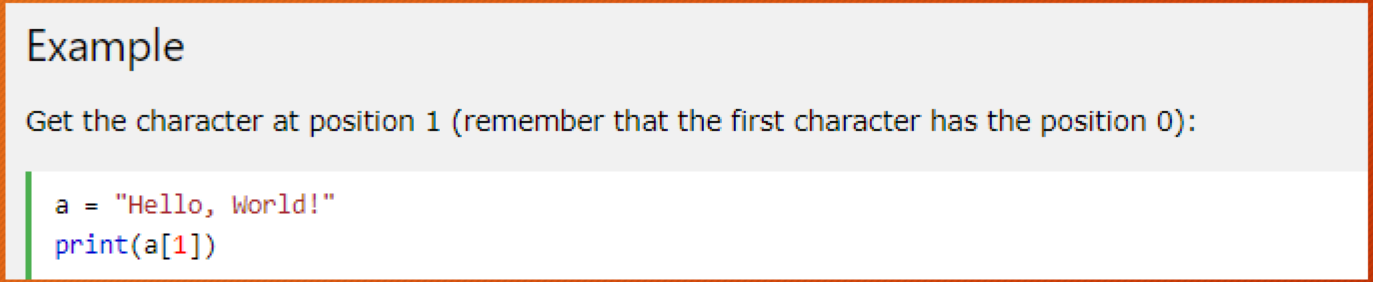
```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

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.

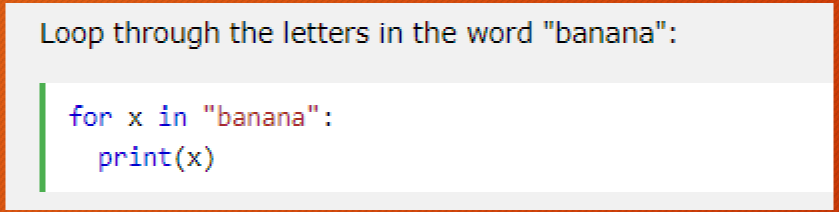


Example

Get the character at position 1 (remember that the first character has the position 0):

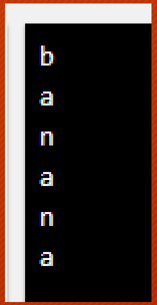
```python
a = "Hello, World!"
print(a[1])
```

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



Loop through the letters in the word "banana":

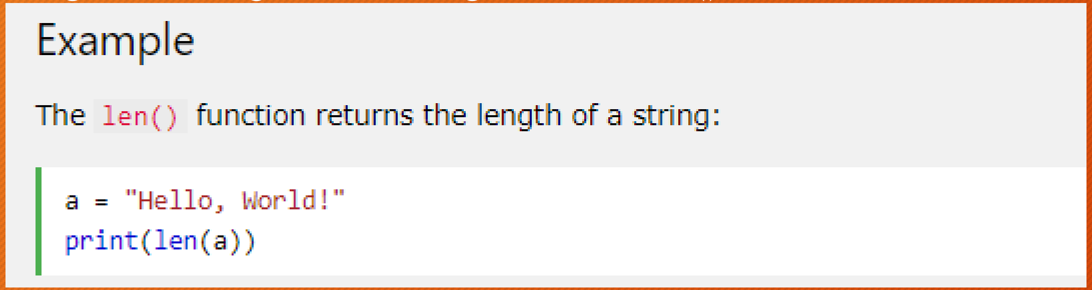
```python
for x in "banana":
  print(x)
```



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

Strings are Arrays

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

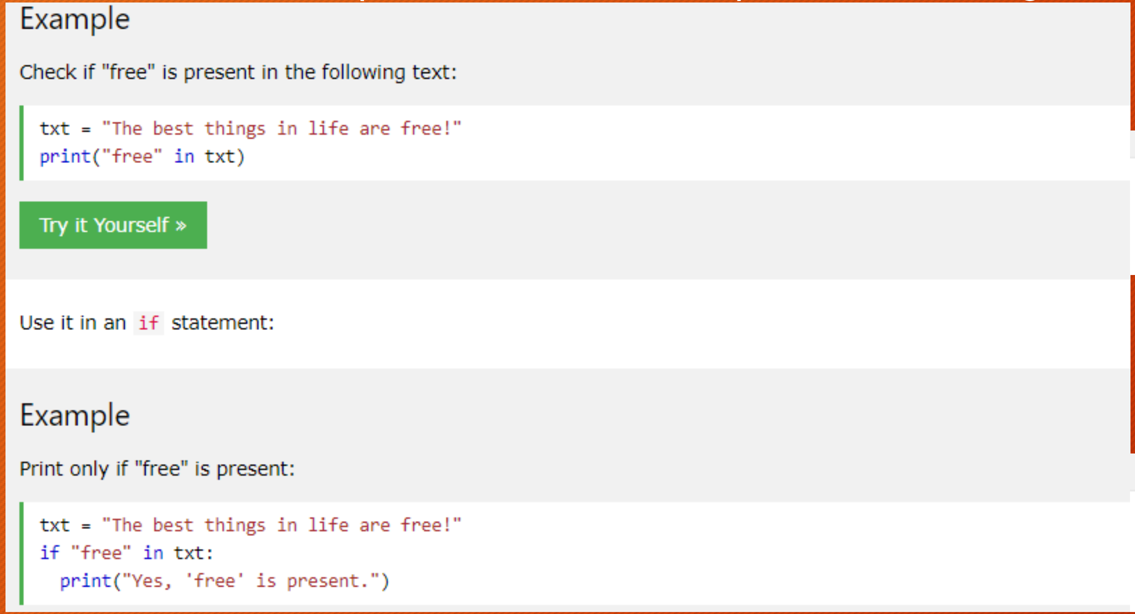


Example

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

```python
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:

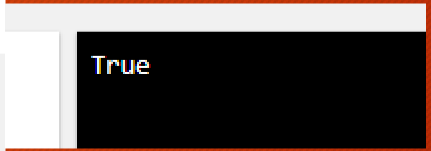
```python
txt = "The best things in life are free!"
print("free" in txt)
```

Use it in an `if` statement:

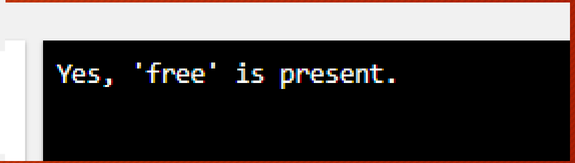
Example

Print only if "free" is present:

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



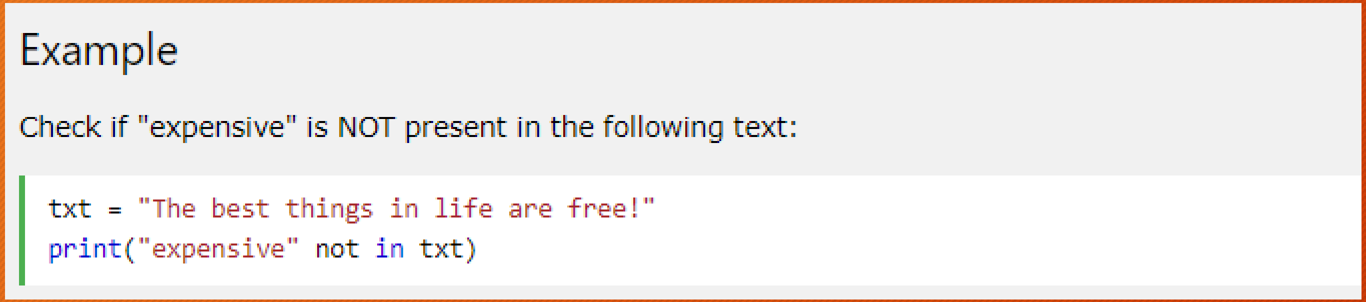
```
True
```



```
Yes, 'free' is present.
```

Strings are Arrays

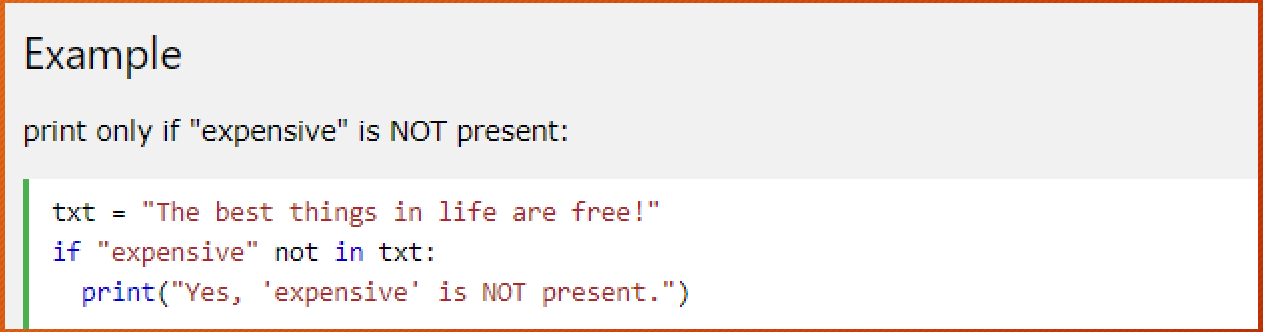
Check if NOT
To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.



Example

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

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

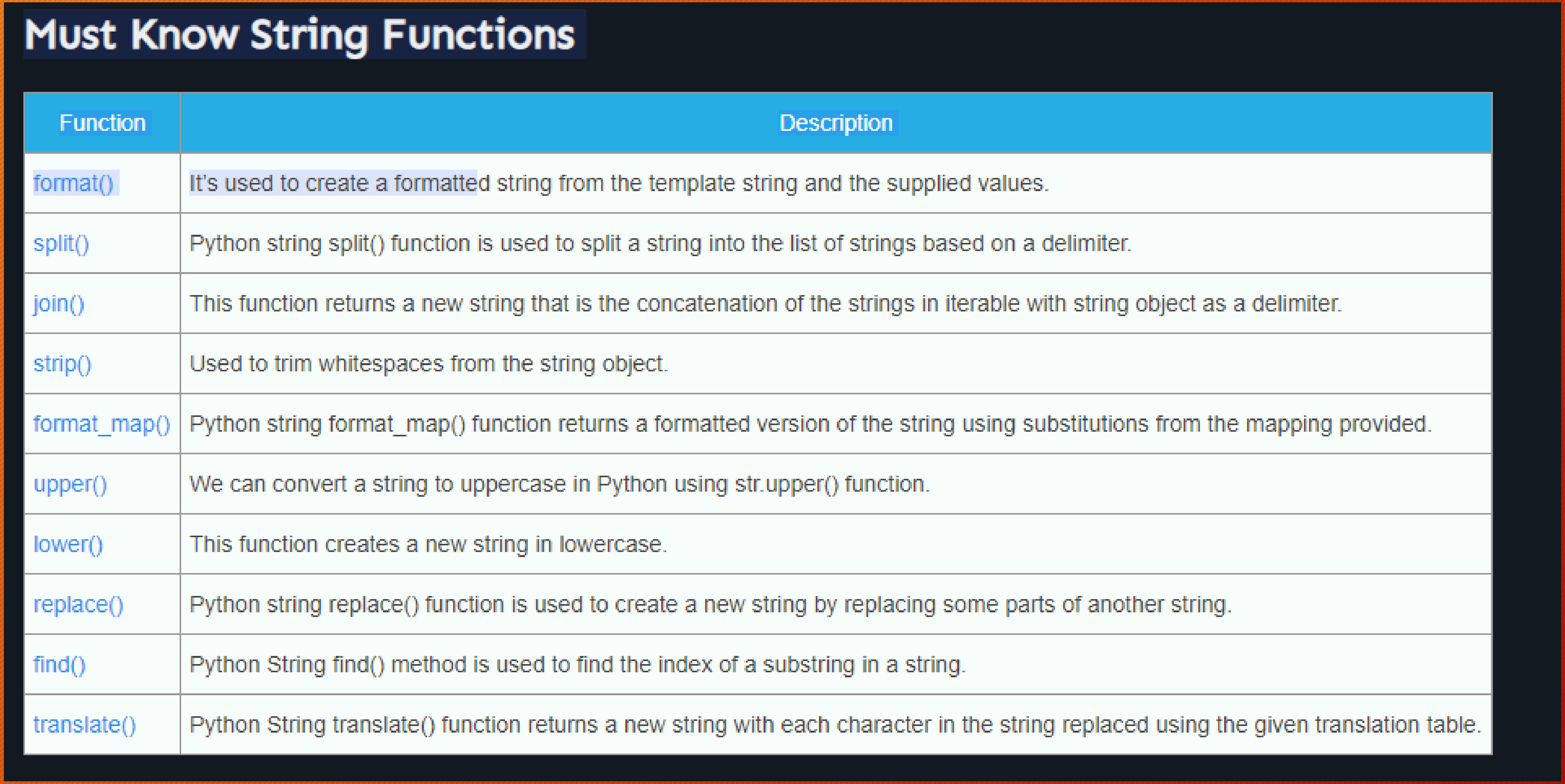


Example

print only if "expensive" is NOT present:

```python
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("Yes, 'expensive' is NOT present.")
```

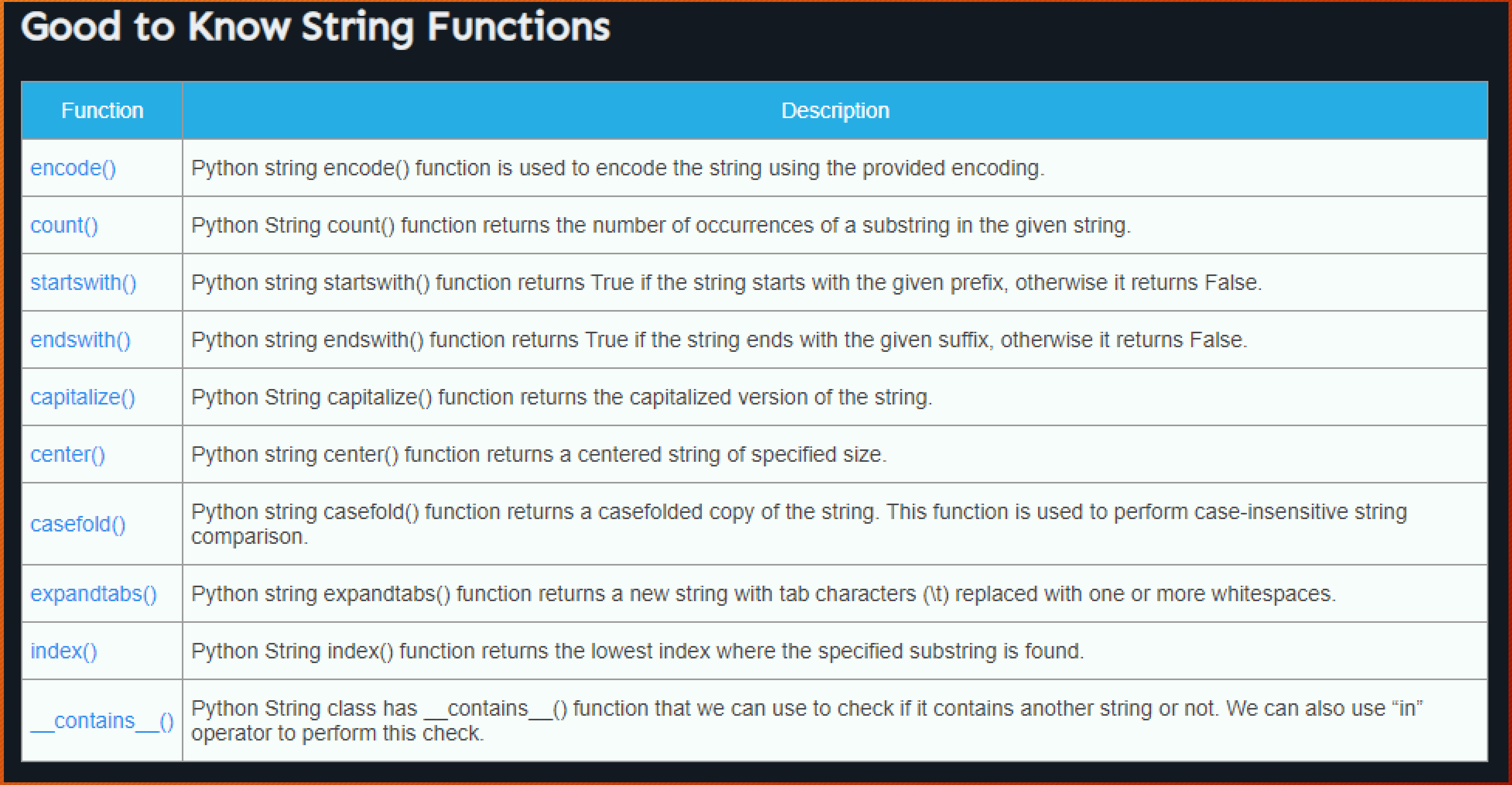
Strings



Must Know String Functions

Function	Description
format()	It's used to create a formatted string from the template string and the supplied values.
split()	Python string split() function is used to split a string into the list of strings based on a delimiter.
join()	This function returns a new string that is the concatenation of the strings in iterable with string object as a delimiter.
strip()	Used to trim whitespaces from the string object.
format_map()	Python string format_map() function returns a formatted version of the string using substitutions from the mapping provided.
upper()	We can convert a string to uppercase in Python using str.upper() function.
lower()	This function creates a new string in lowercase.
replace()	Python string replace() function is used to create a new string by replacing some parts of another string.
find()	Python String find() method is used to find the index of a substring in a string.
translate()	Python String translate() function returns a new string with each character in the string replaced using the given translation table.

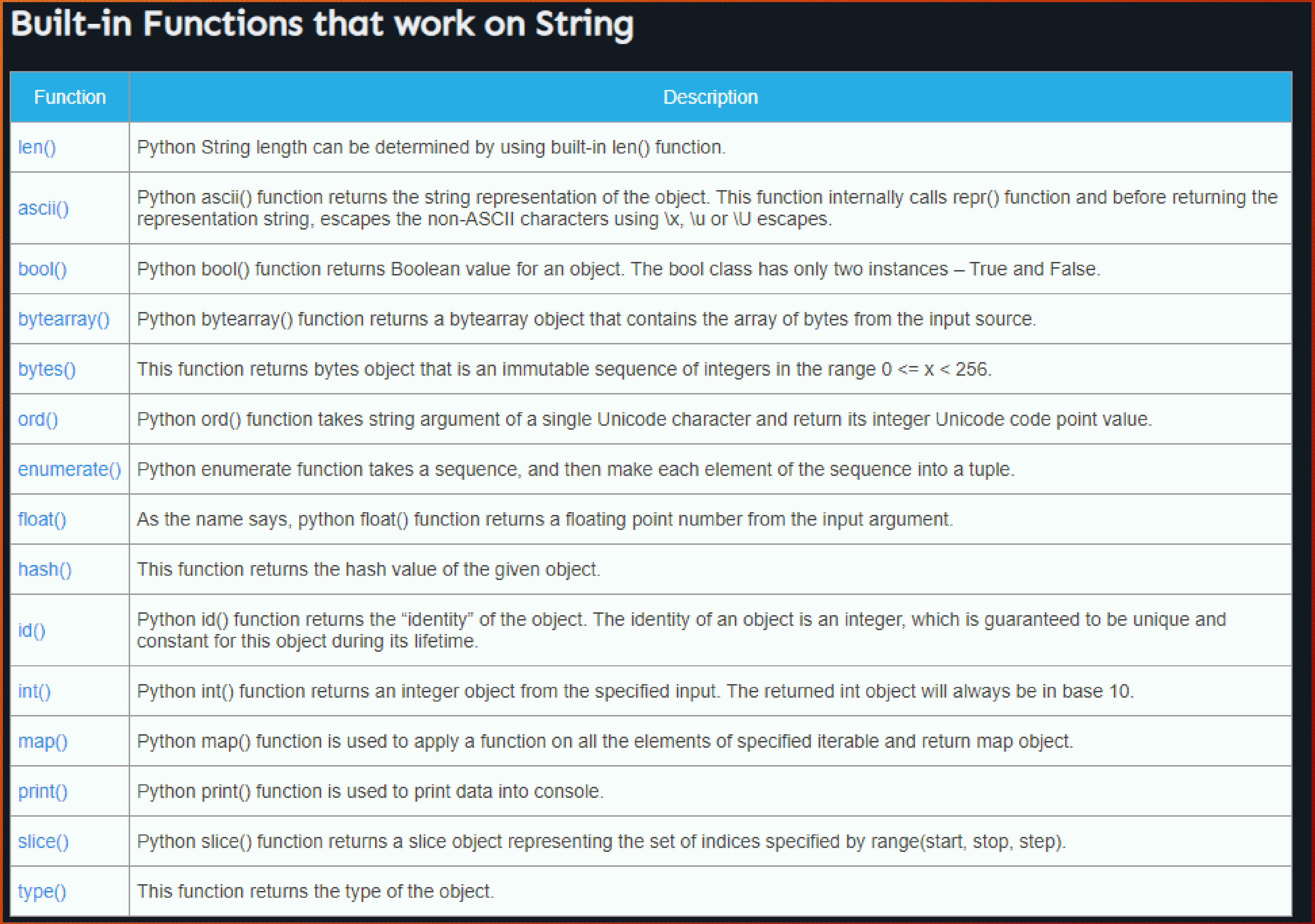
Strings



Good to Know String Functions

Function	Description
encode()	Python string encode() function is used to encode the string using the provided encoding.
count()	Python String count() function returns the number of occurrences of a substring in the given string.
startswith()	Python string startswith() function returns True if the string starts with the given prefix, otherwise it returns False.
endswith()	Python string endswith() function returns True if the string ends with the given suffix, otherwise it returns False.
capitalize()	Python String capitalize() function returns the capitalized version of the string.
center()	Python string center() function returns a centered string of specified size.
casefold()	Python string casefold() function returns a casefolded copy of the string. This function is used to perform case-insensitive string comparison.
expandtabs()	Python string expandtabs() function returns a new string with tab characters (\t) replaced with one or more whitespaces.
index()	Python String index() function returns the lowest index where the specified substring is found.
__contains__()	Python String class has __contains__() function that we can use to check if it contains another string or not. We can also use “in” operator to perform this check.

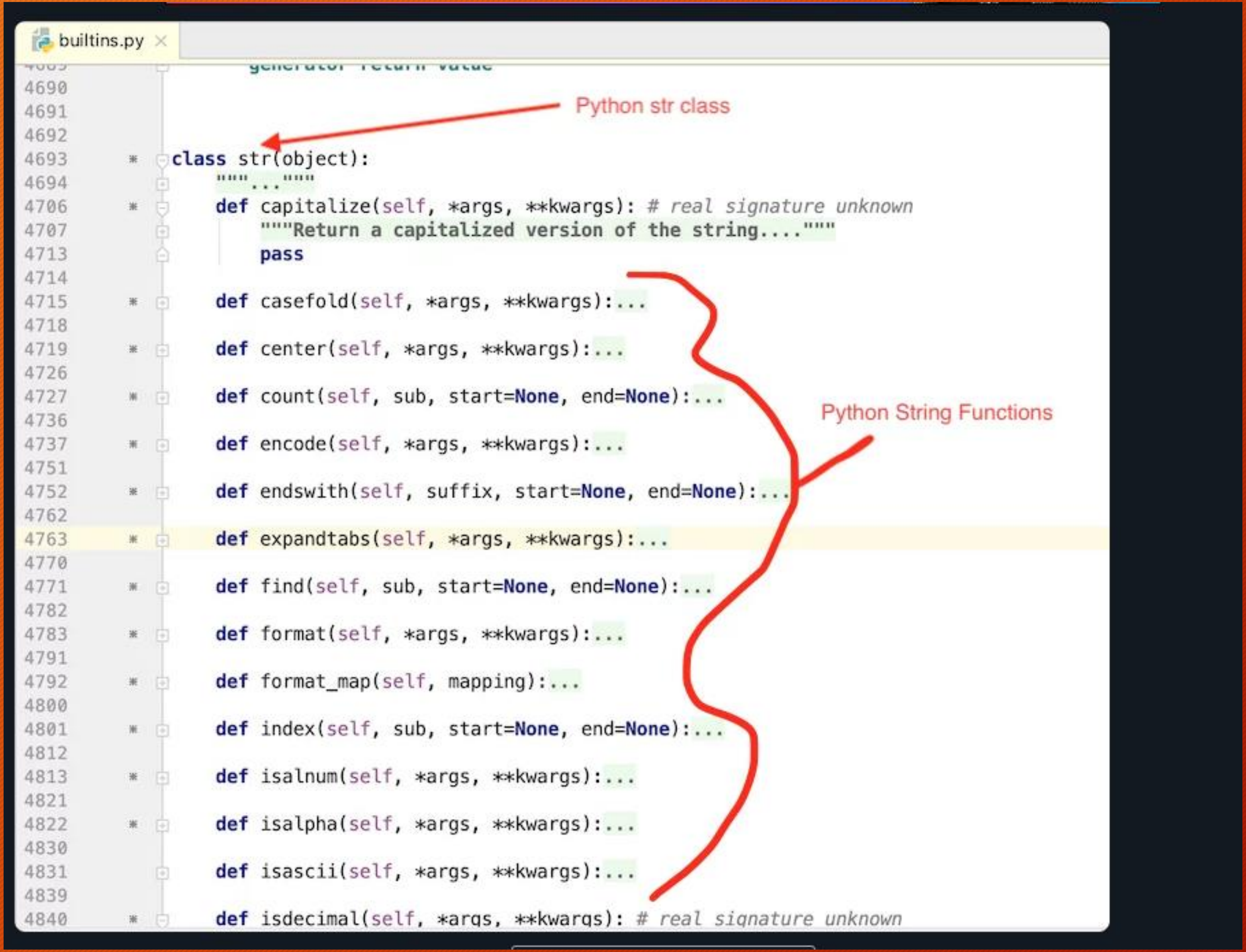
Strings



Built-in Functions that work on String

Function	Description
len()	Python String length can be determined by using built-in len() function.
ascii()	Python ascii() function returns the string representation of the object. This function internally calls repr() function and before returning the representation string, escapes the non-ASCII characters using \x, \u or \U escapes.
bool()	Python bool() function returns Boolean value for an object. The bool class has only two instances – True and False.
bytearray()	Python bytearray() function returns a bytearray object that contains the array of bytes from the input source.
bytes()	This function returns bytes object that is an immutable sequence of integers in the range 0 <= x < 256.
ord()	Python ord() function takes string argument of a single Unicode character and return its integer Unicode code point value.
enumerate()	Python enumerate function takes a sequence, and then make each element of the sequence into a tuple.
float()	As the name says, python float() function returns a floating point number from the input argument.
hash()	This function returns the hash value of the given object.
id()	Python id() function returns the “identity” of the object. The identity of an object is an integer, which is guaranteed to be unique and constant for this object during its lifetime.
int()	Python int() function returns an integer object from the specified input. The returned int object will always be in base 10.
map()	Python map() function is used to apply a function on all the elements of specified iterable and return map object.
print()	Python print() function is used to print data into console.
slice()	Python slice() function returns a slice object representing the set of indices specified by range(start, stop, step).
type()	This function returns the type of the object.

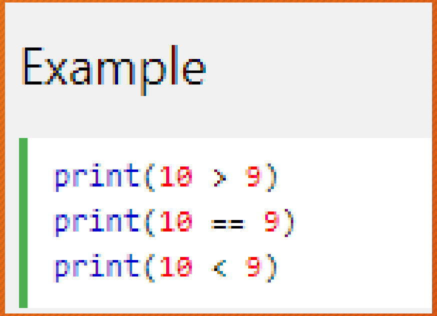
Do I Need to remember all of them ???



```
        generator return value


class str(object):
    """..."""
    def capitalize(self, *args, **kwargs): # real signature unknown
        """Return a capitalized version of the string...."""
        pass

    def casefold(self, *args, **kwargs):...

    def center(self, *args, **kwargs):...

    def count(self, sub, start=None, end=None):...

    def encode(self, *args, **kwargs):...

    def endswith(self, suffix, start=None, end=None):...

    def expandtabs(self, *args, **kwargs):...

    def find(self, sub, start=None, end=None):...

    def format(self, *args, **kwargs):...

    def format_map(self, mapping):...

    def index(self, sub, start=None, end=None):...

    def isalnum(self, *args, **kwargs):...

    def isalpha(self, *args, **kwargs):...

    def isascii(self, *args, **kwargs):...

    def isdecimal(self, *args, **kwargs): # real signature unknown
```

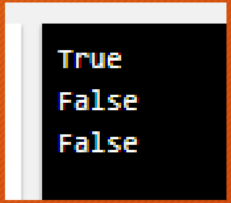
Python Boolean

Booleans represent one of two values: True or False.
Boolean Values
In programming you often need to know if an expression is True or False.
You can evaluate any expression in Python, and get one of two answers, True or False.
When you compare two values, the expression is evaluated, and Python returns the Boolean answer:



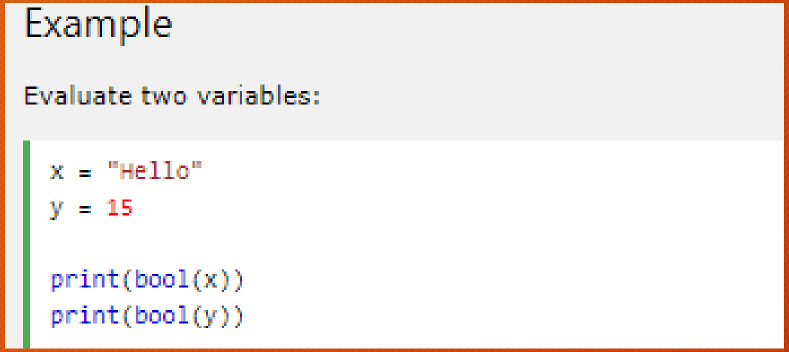
Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```



```
True
False
False
```

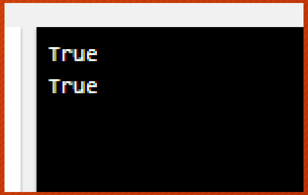
Evaluate Values and Variables
The bool() function allows you to evaluate any value, and give you True or False in return,



Example

Evaluate two variables:

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

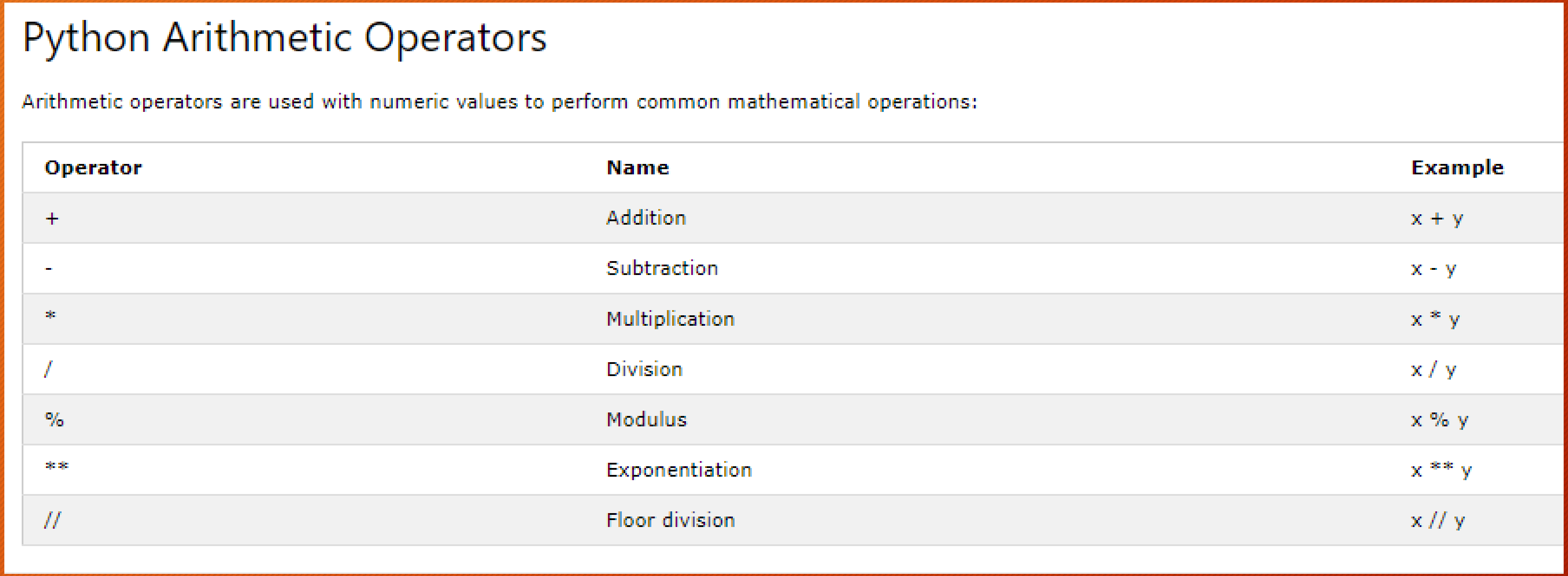


```
True
True
```

Python Operators

Python Operators
Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

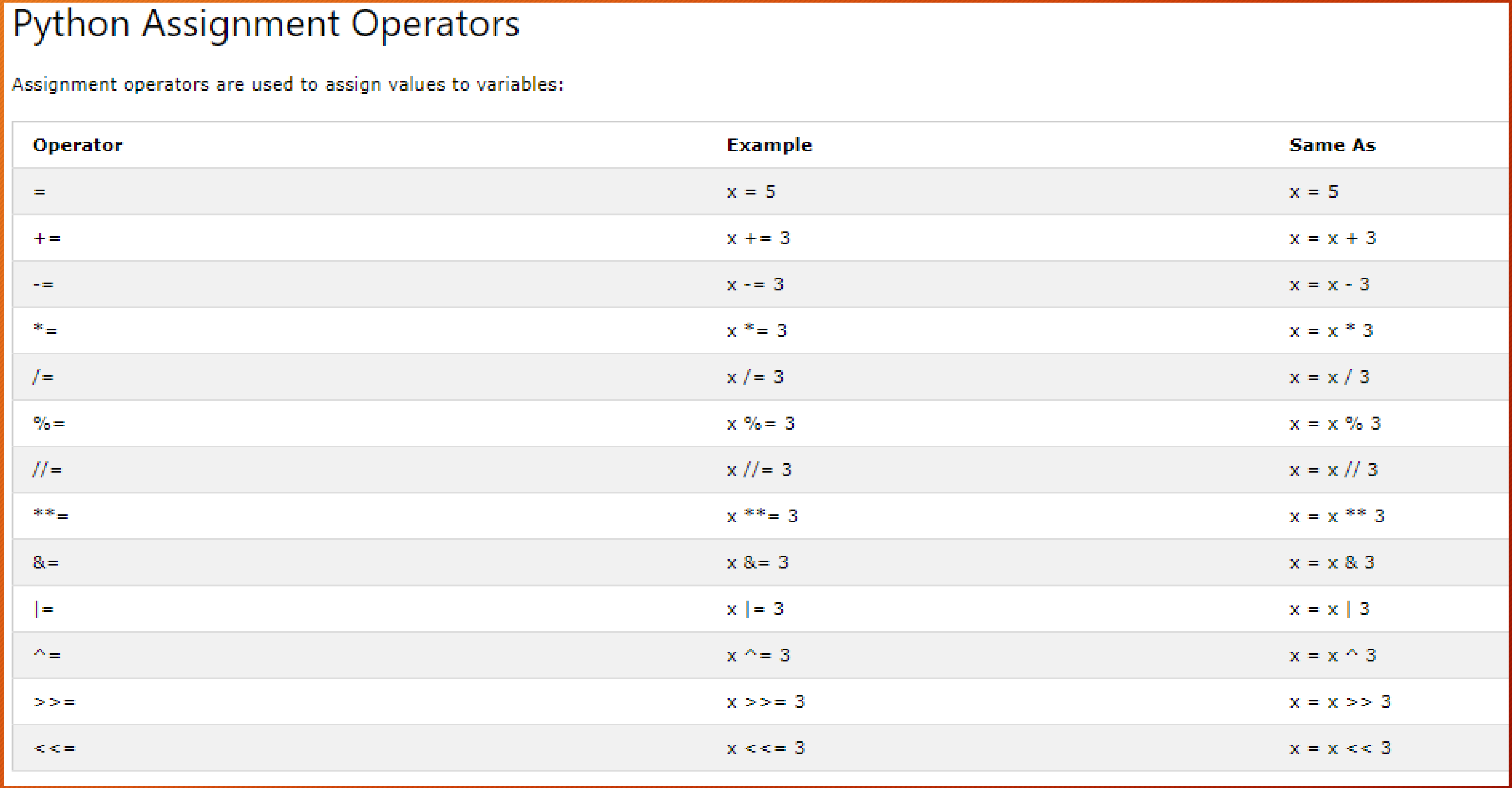


Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Operators



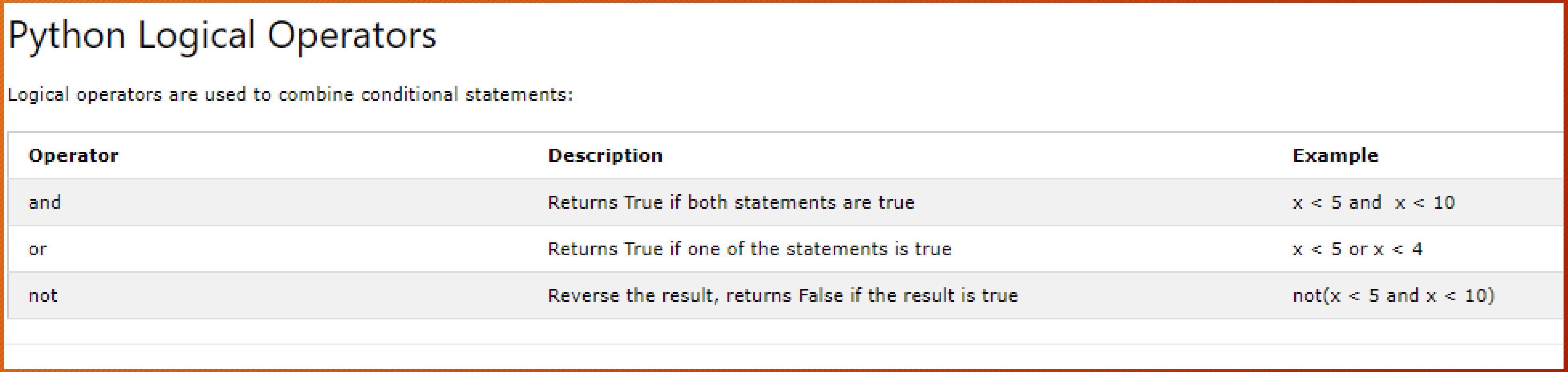
Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Operators

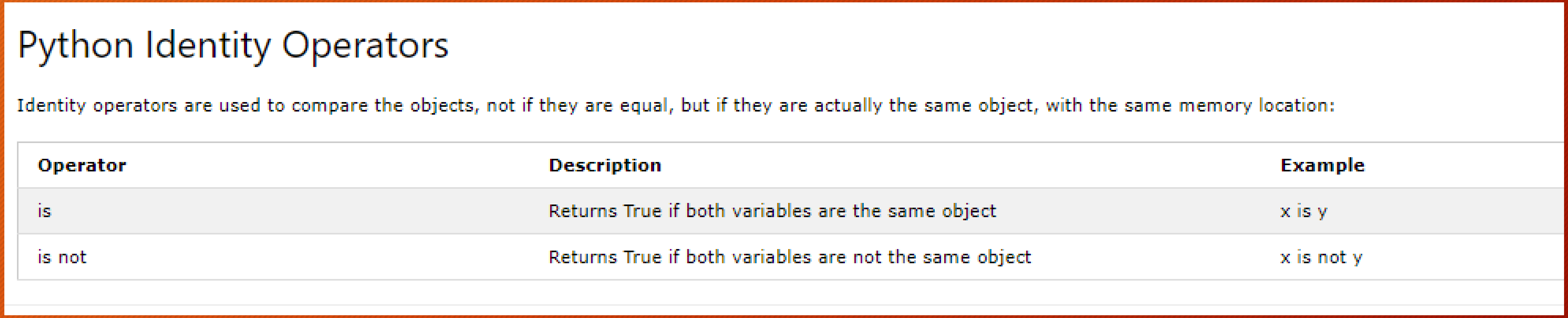
Python Operators



Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)



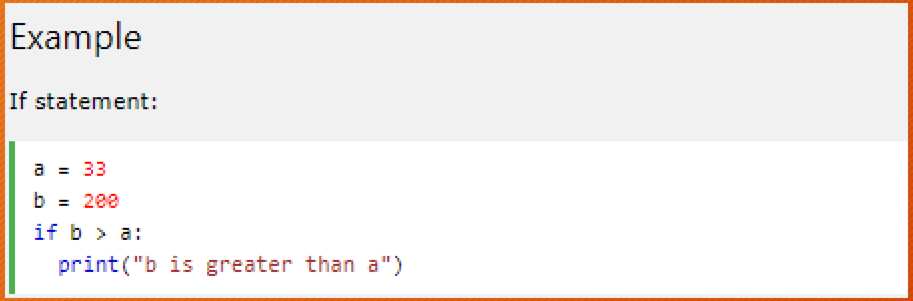
Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python If ... Else

If statement:

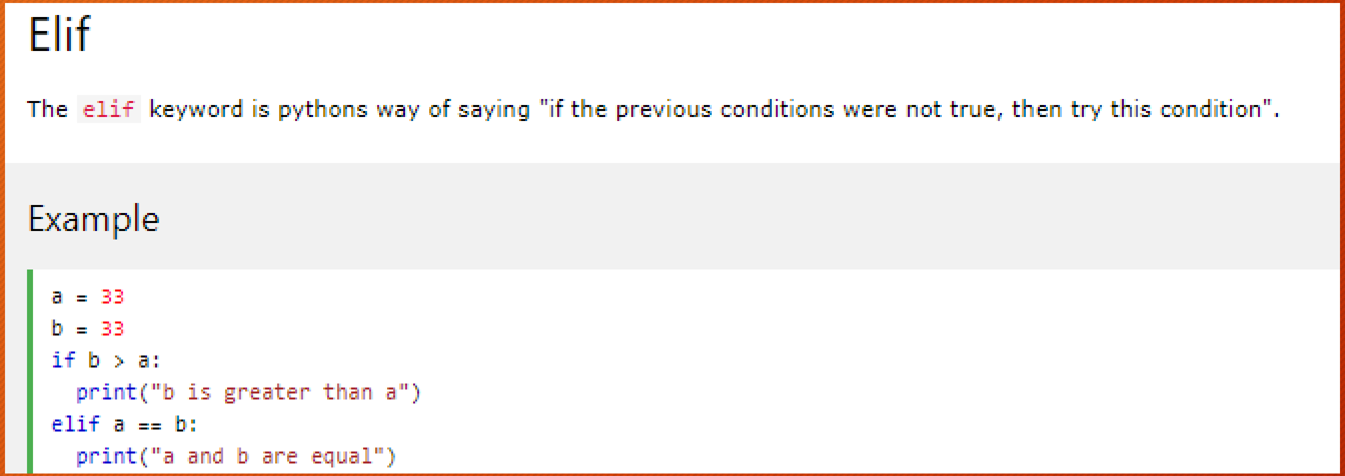


Example

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Java programming language use curly-brackets for this purpose.



Elif

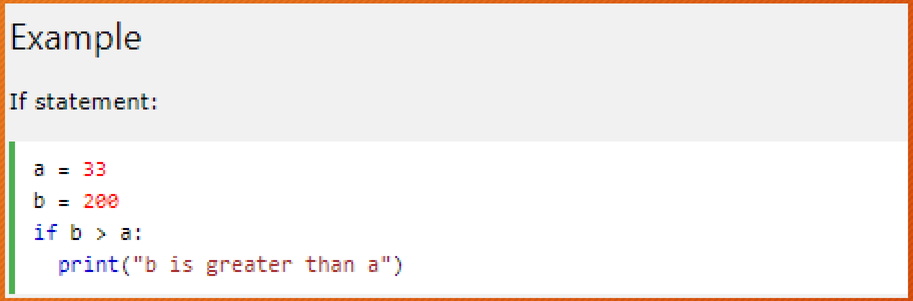
The `elif` keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Python If ... Else

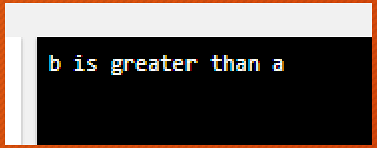
If statement:



Example

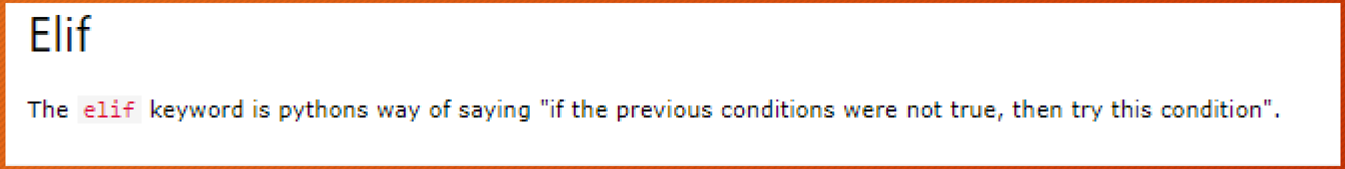
If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```



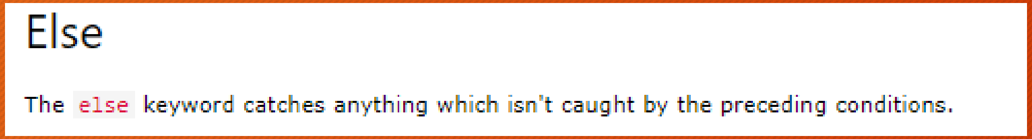
```
b is greater than a
```

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Java programming language use curly-brackets for this purpose.



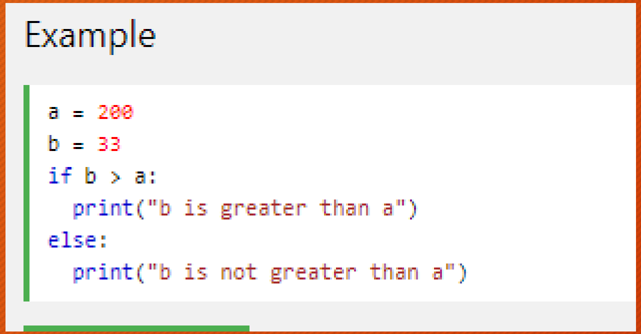
Elif

The `elif` keyword is pythons way of saying "if the previous conditions were not true, then try this condition".



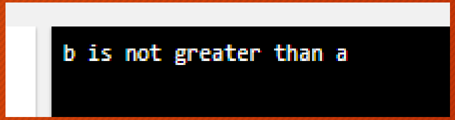
Else

The `else` keyword catches anything which isn't caught by the preceding conditions.



Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

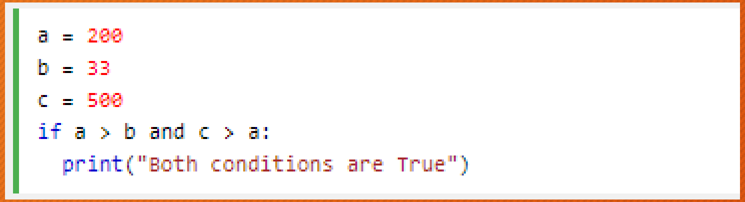


```
b is not greater than a
```

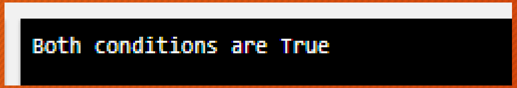
Python If ... Else

And

The and keyword is a logical operator, and is used to combine conditional statements:



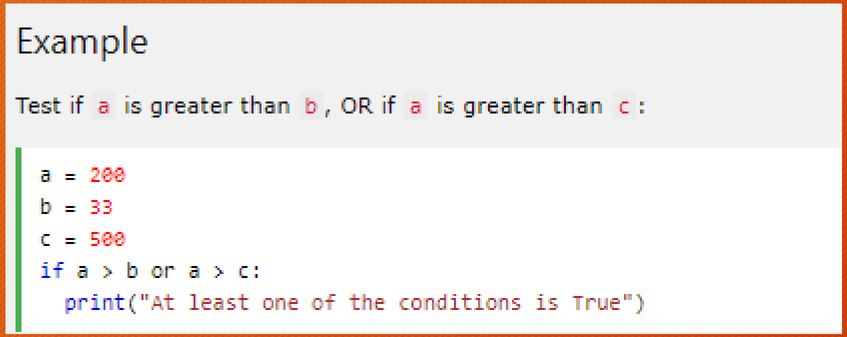
```python
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```



```
Both conditions are True
```

Or

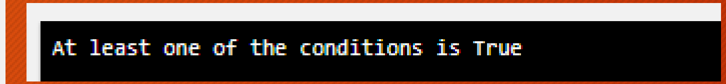
The or keyword is a logical operator, and is used to combine conditional statements:



Example

Test if a is greater than b, OR if a is greater than c:

```python
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

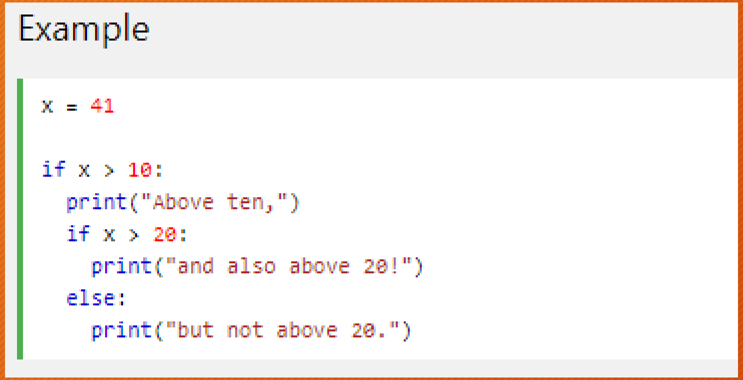


```
At least one of the conditions is True
```

Python If ... Else

Nested If

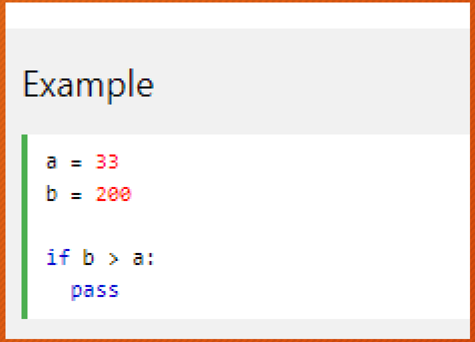
You can have if statements inside if statements, this is called nested if statements.


Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

The pass Statement

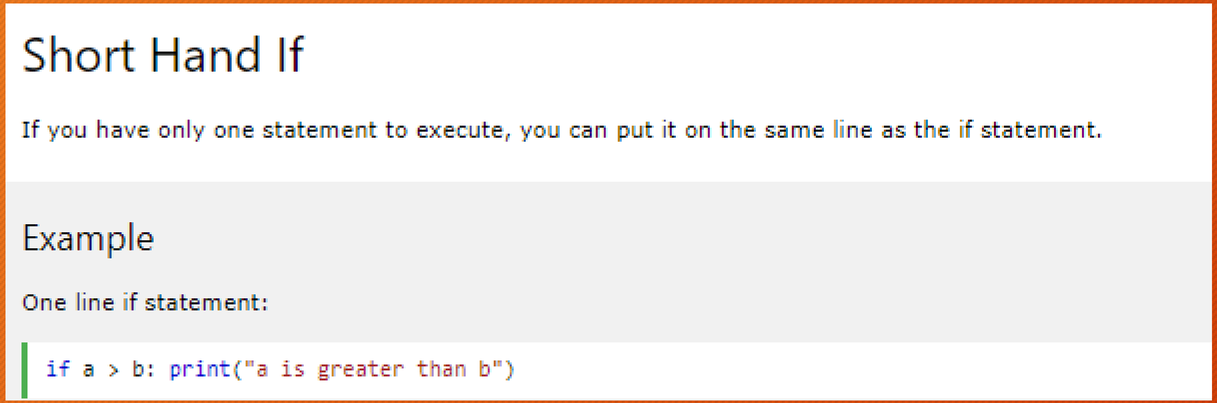
if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.


Example

```
a = 33
b = 200

if b > a:
  pass
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.



Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

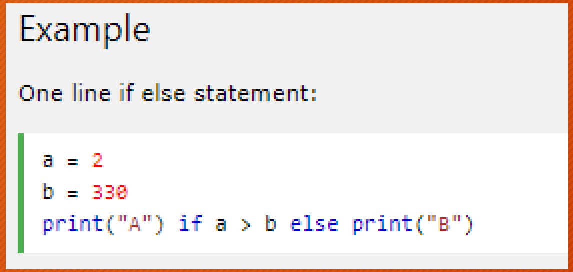
Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:



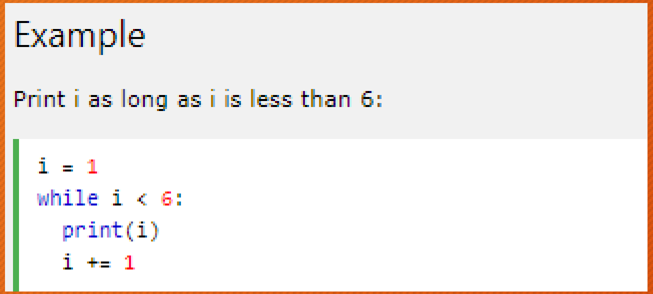
Example

One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Python While Loops

The while Loop
With the while loop we can execute a set of statements as long as a condition is true.

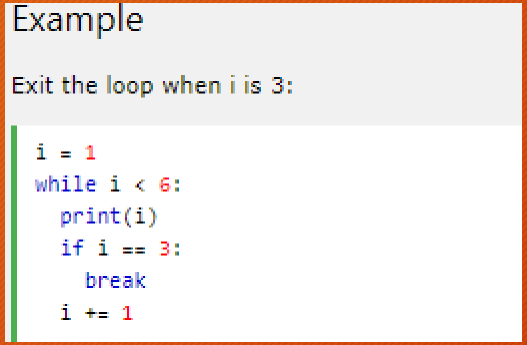


Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

The while Loop
With the while loop we can execute a set of statements as long as a condition is true.



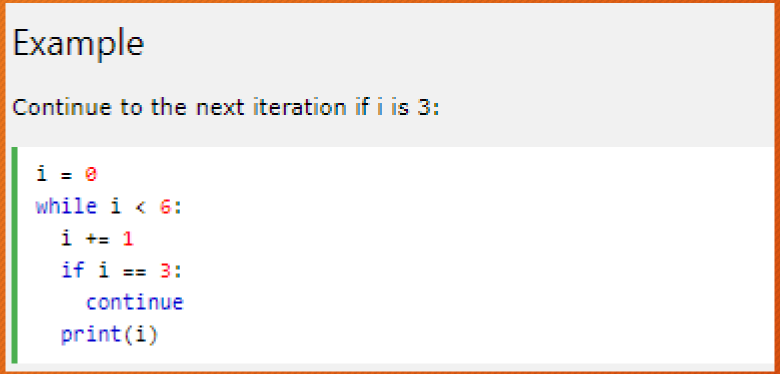
Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Python While Loops

The continue Statement
With the continue statement we can stop the current iteration, and continue with the next:

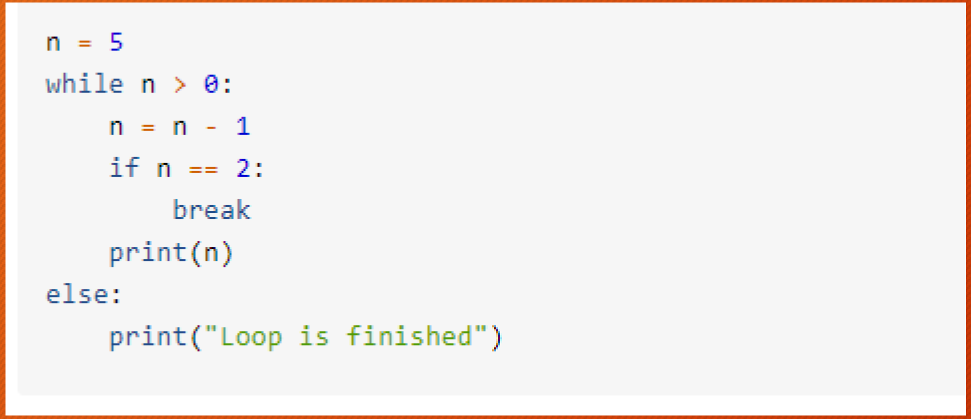


Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

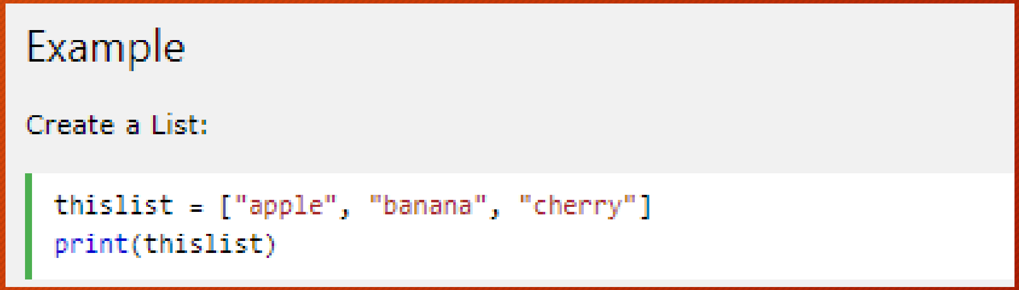
while-loop-else-clause, which is unique to Python. The else-block is only executed if the while-loop is exhausted.



```
n = 5
while n > 0:
    n = n - 1
    if n == 2:
        break
    print(n)
else:
    print("Loop is finished")
```

Python Lists

List
Lists are used to store multiple items in a single variable.
Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
Lists are created using square brackets:



Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List items are ordered, changeable, and allow duplicate values.
List items are indexed, the first item has index [0], the second item has index [1] etc.\

Ordered
When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

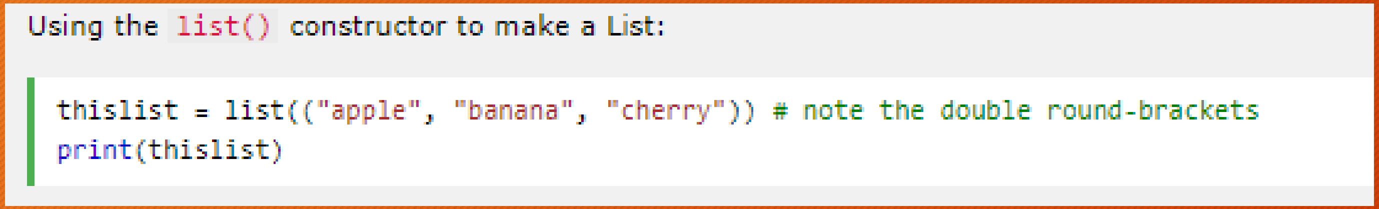
Note: There are some list methods that will change the order, but in general: the order of the items will not change.

Changeable
The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates
Since lists are indexed, lists can have items with the same value.

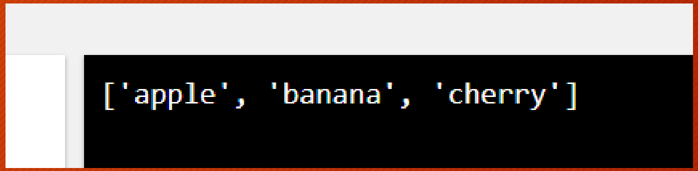
Python Lists

The list() Constructor
It is also possible to use the list() constructor when creating a new list.

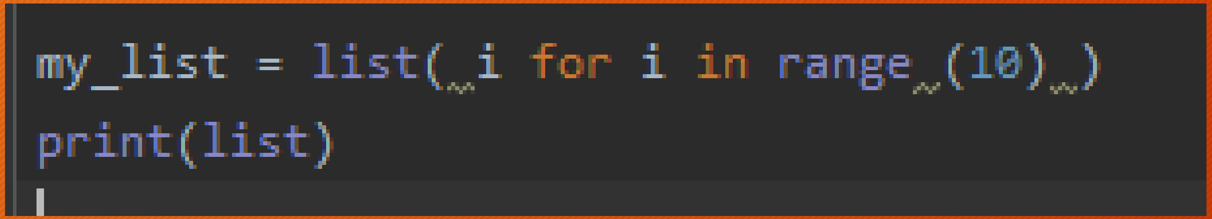


Using the `list()` constructor to make a List:

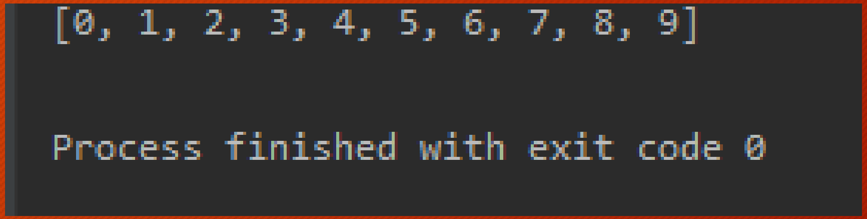
```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```



```
['apple', 'banana', 'cherry']
```



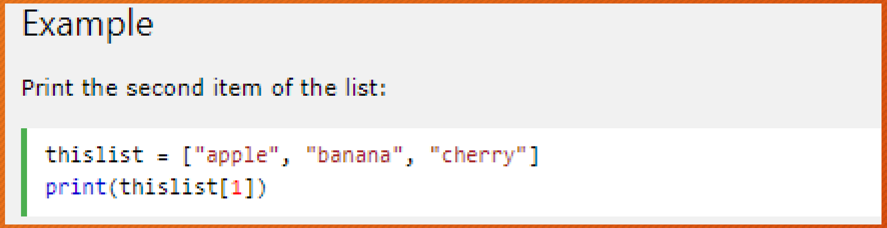
```
my_list = list( i for i in range (10) )
print(list)
```



```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Process finished with exit code 0
```

Python - Access List Items

Access Items
List items are indexed and you can access them by referring to the index number:

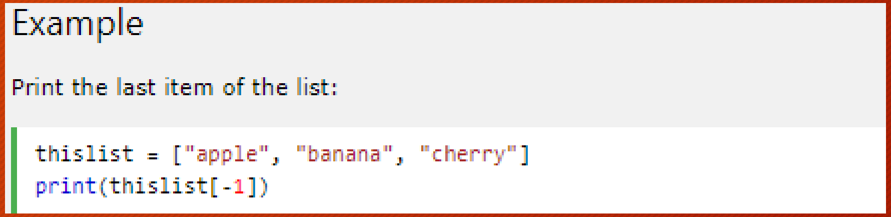

Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Negative Indexing
Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

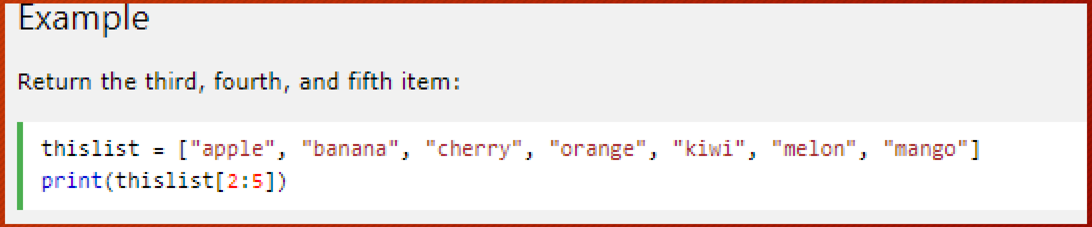

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Negative Indexing
Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.


Example

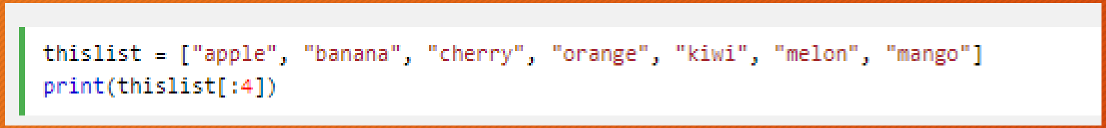
Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

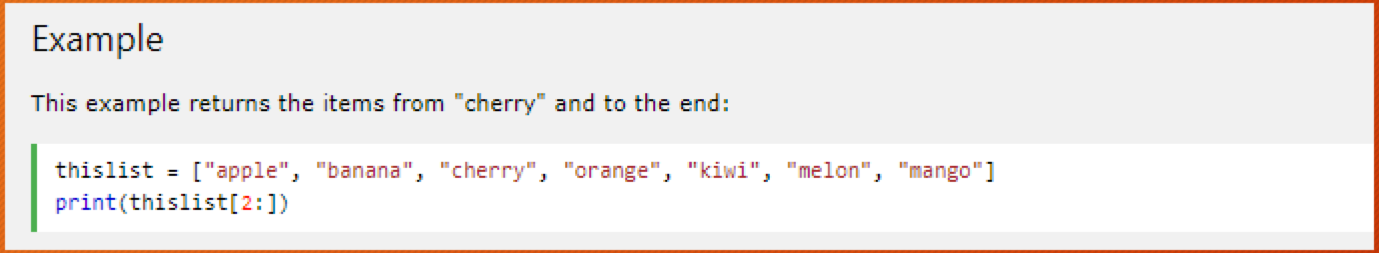
Note: The search will start at index 2 (included) and end at index 5 (not included).

Python - Access List Items

Example
This example returns the items from the beginning to, but NOT included, "kiwi":



```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

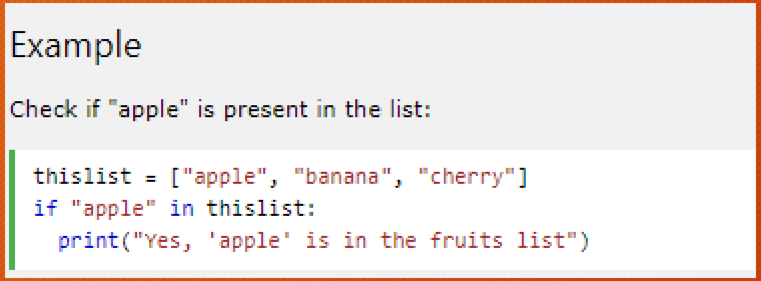


Example

This example returns the items from "cherry" and to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Check if Item Exists
To determine if a specified item is present in a list use the in keyword:

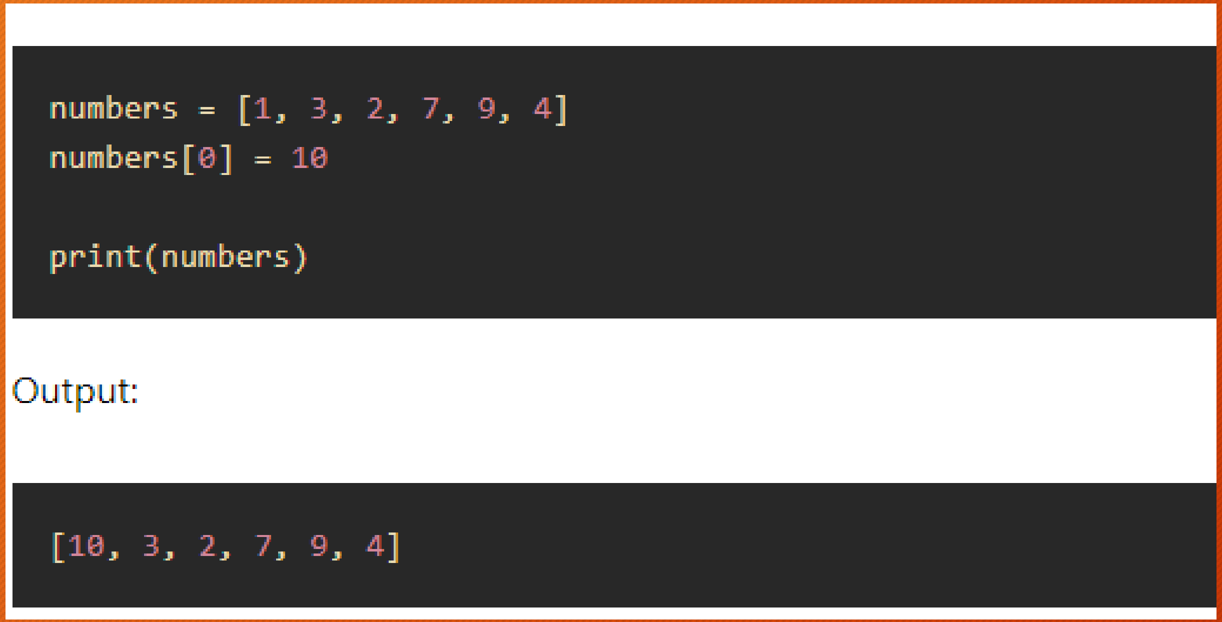


Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

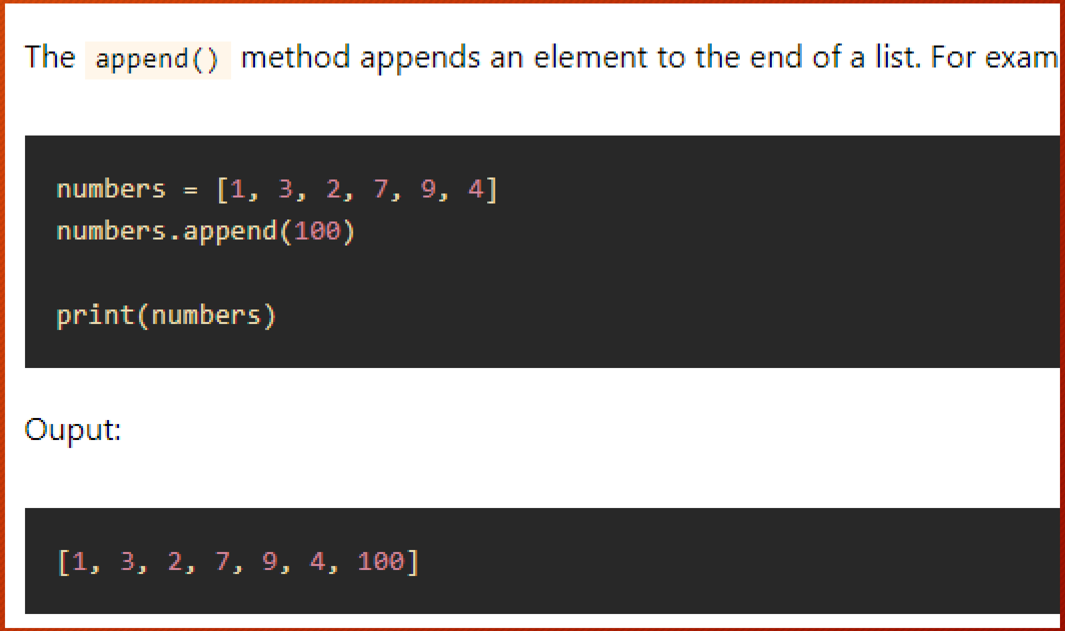
Modifying elements in a list



```
numbers = [1, 3, 2, 7, 9, 4]
numbers[0] = 10

print(numbers)
```

Output:

```
[10, 3, 2, 7, 9, 4]
```



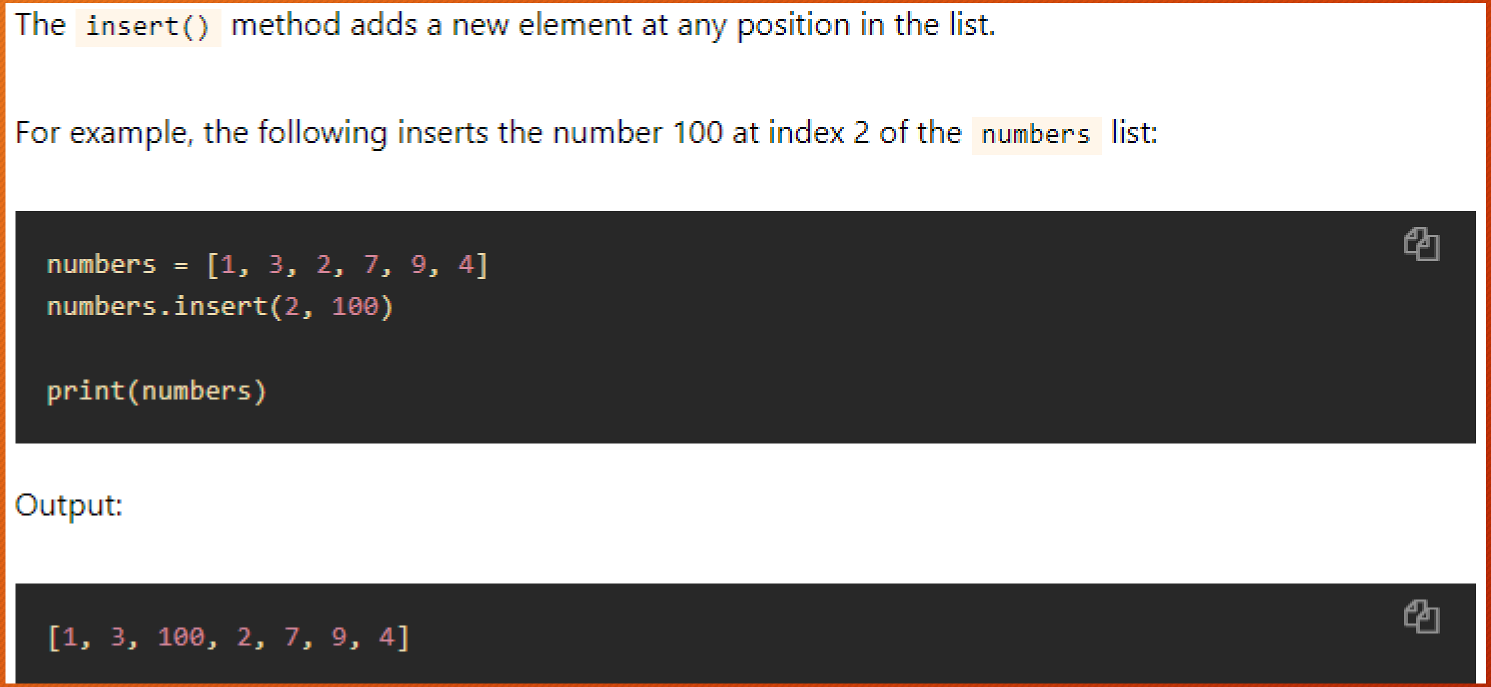
The `append()` method appends an element to the end of a list. For exam

```
numbers = [1, 3, 2, 7, 9, 4]
numbers.append(100)

print(numbers)
```

Ouput:

```
[1, 3, 2, 7, 9, 4, 100]
```

Modifying elements in a list



The `insert()` method adds a new element at any position in the list.

For example, the following inserts the number 100 at index 2 of the `numbers` list:

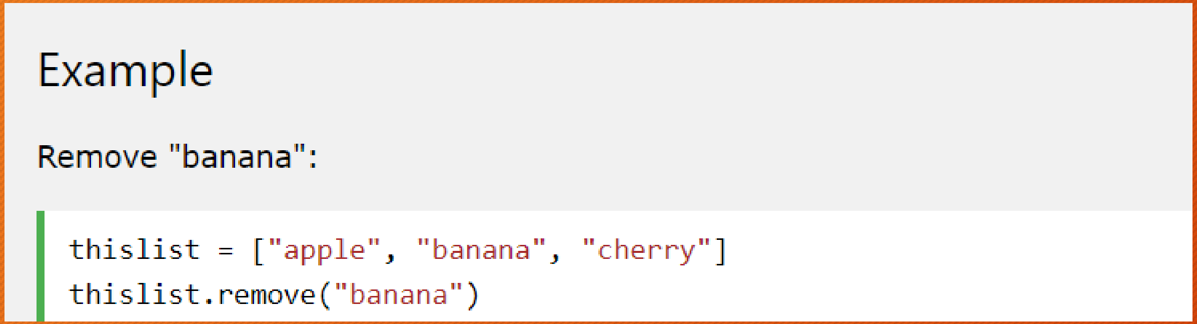
```
numbers = [1, 3, 2, 7, 9, 4]
numbers.insert(2, 100)

print(numbers)
```

Output:

```
[1, 3, 100, 2, 7, 9, 4]
```

Removing elements from a list

The remove() method removes the specified item.

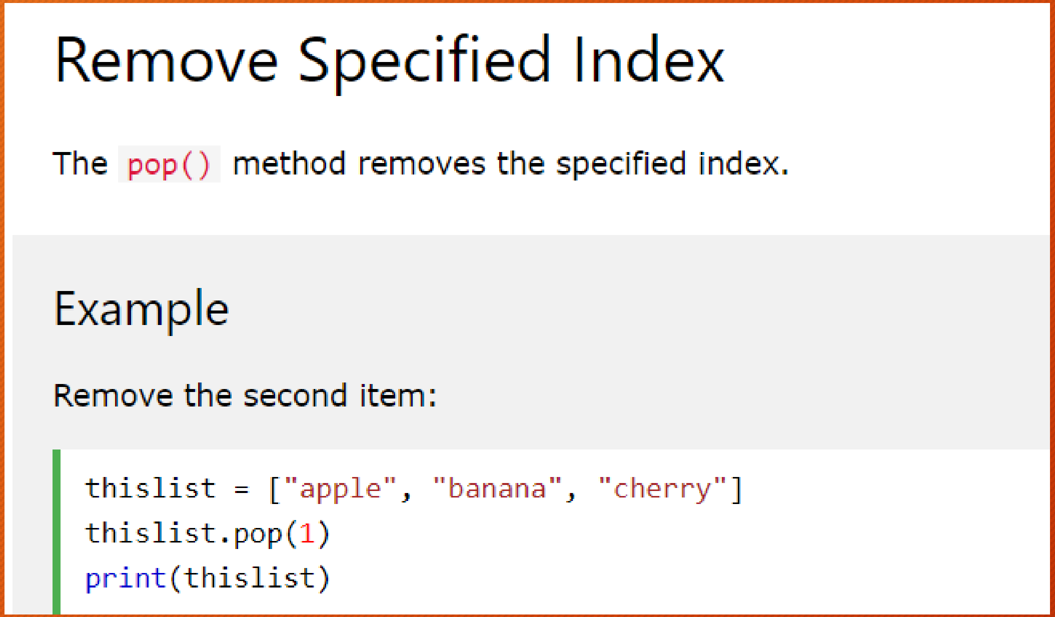


Example

Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
```

The remove() method removes the specified item.



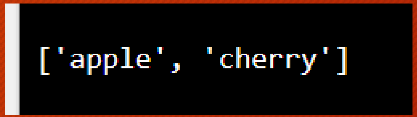
Remove Specified Index

The `pop()` method removes the specified index.

Example

Remove the second item:

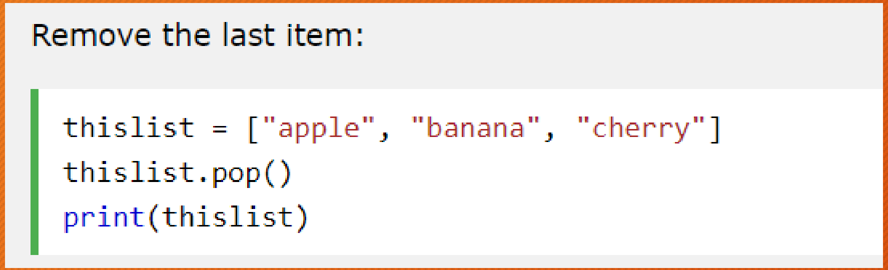
```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```



```
['apple', 'cherry']
```

Removing elements from a list

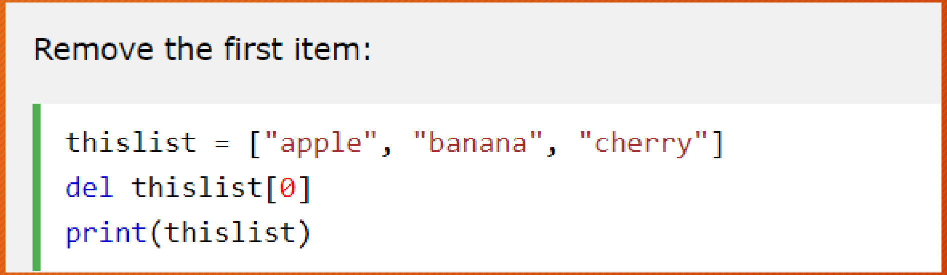
If you do not specify the index, the pop() method removes the last item.



Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

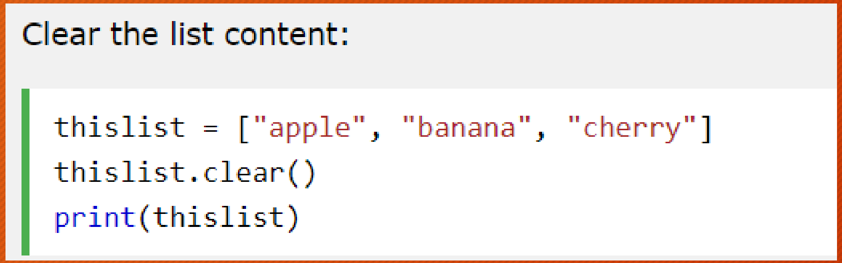
Remove the first item



Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

The clear() method empties the list.
The list still remains, but it has no content.



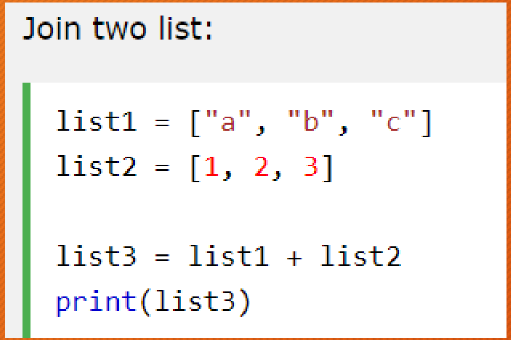
Clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Join Two Lists

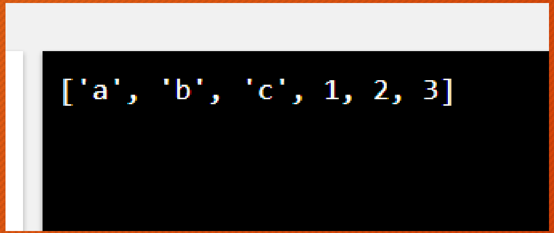
There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.



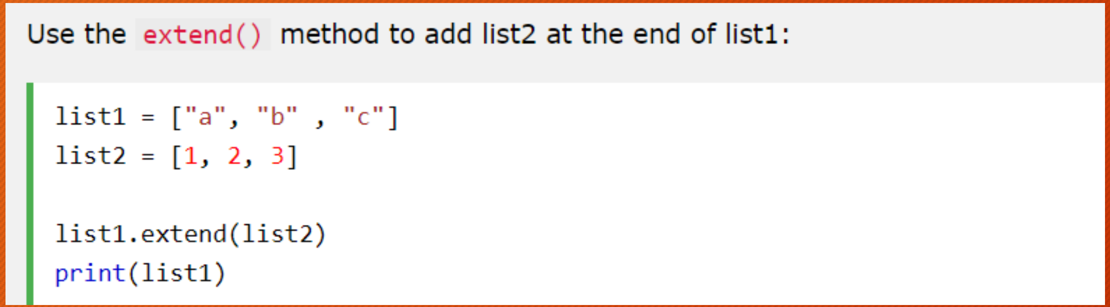
Join two list:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```



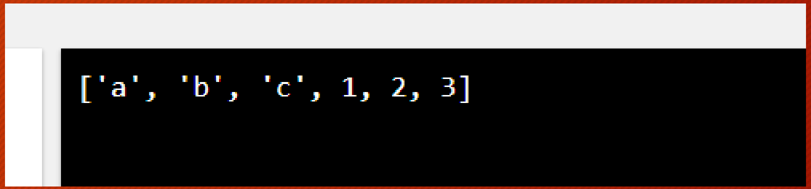
```
['a', 'b', 'c', 1, 2, 3]
```

Use the extend() method to add list2 at the end of list1:



Use the `extend()` method to add list2 at the end of list1:

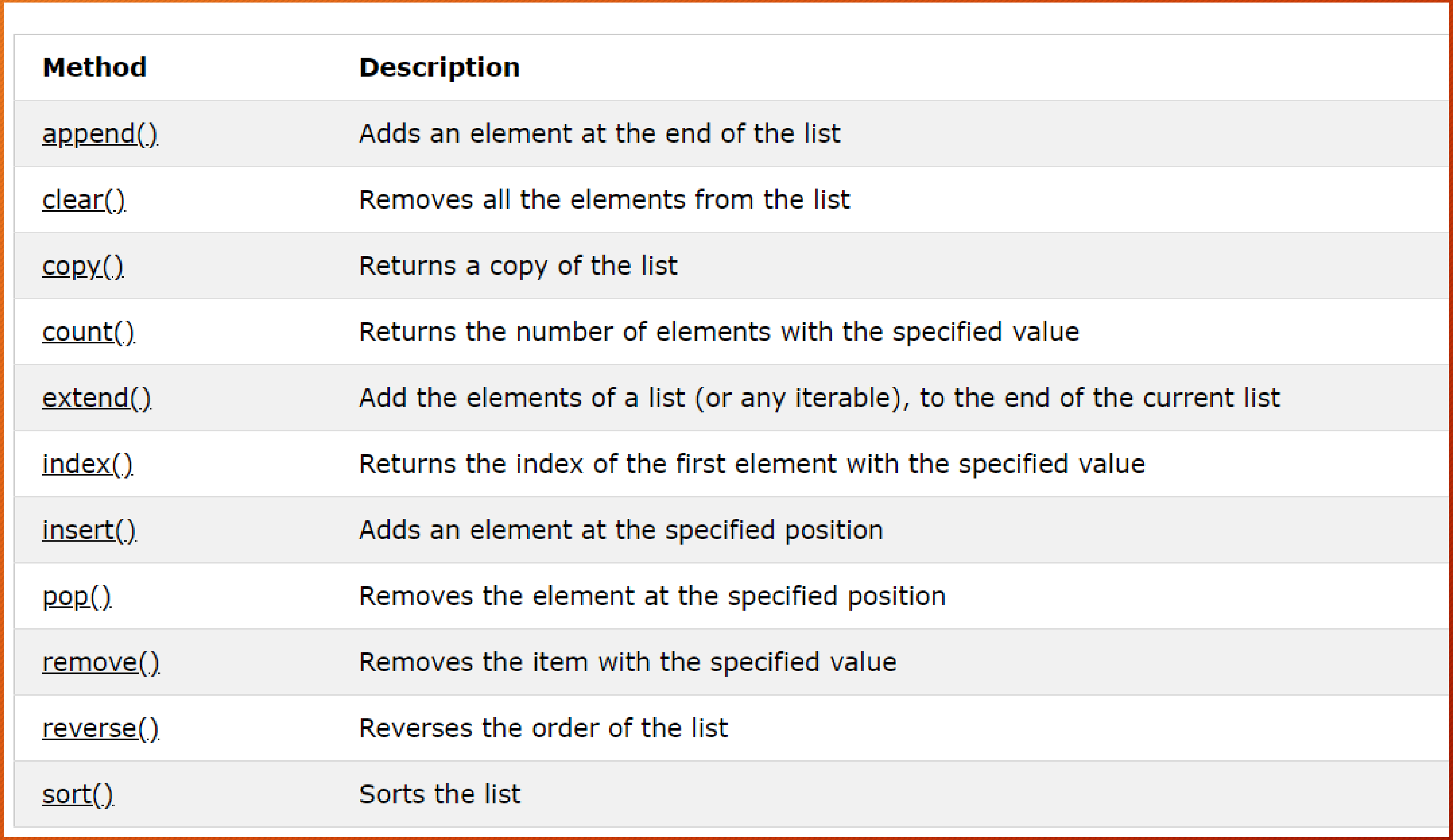
```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```



```
['a', 'b', 'c', 1, 2, 3]
```

List Methods

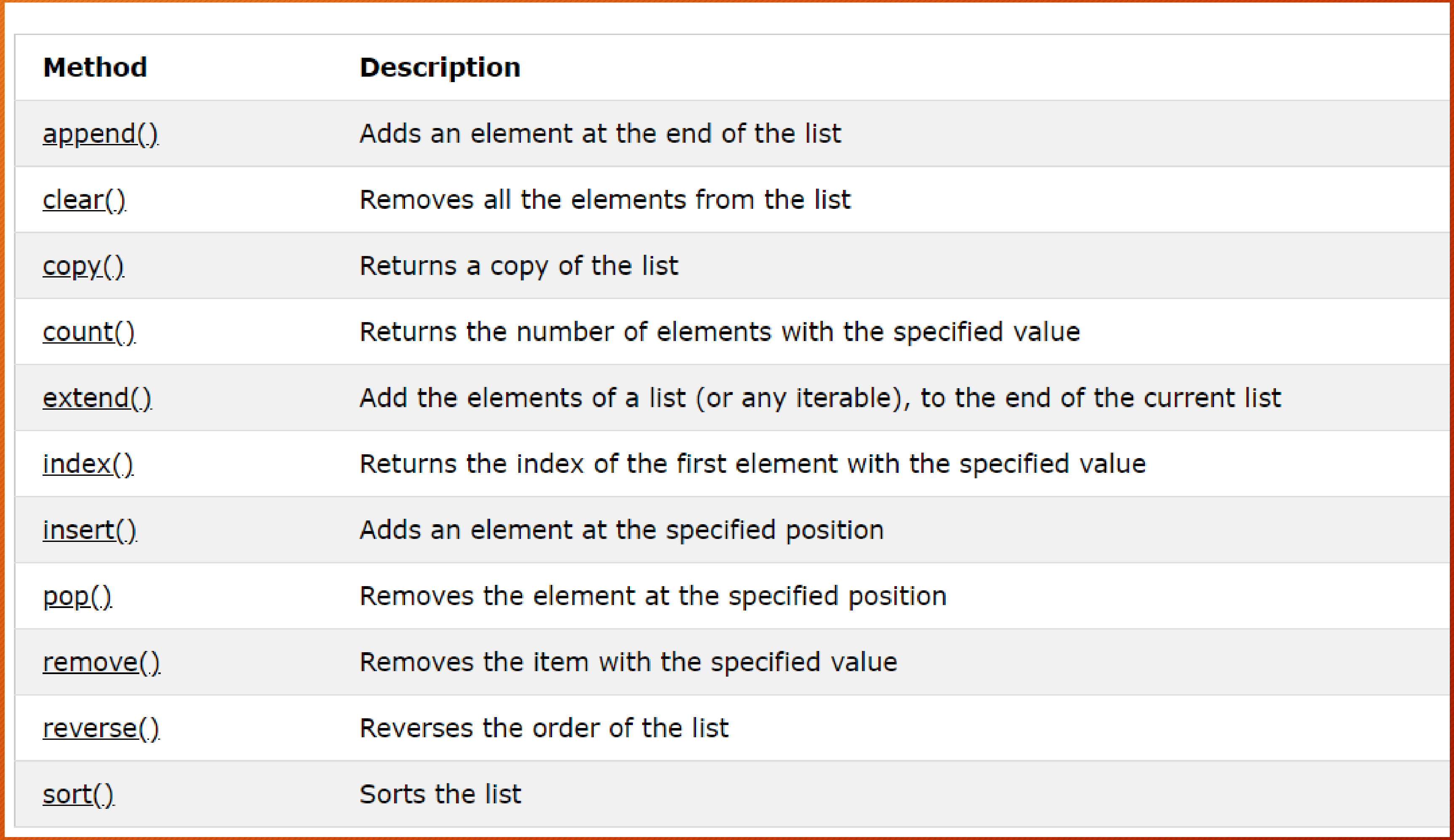
Python has a set of built-in methods that you can use on lists.



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

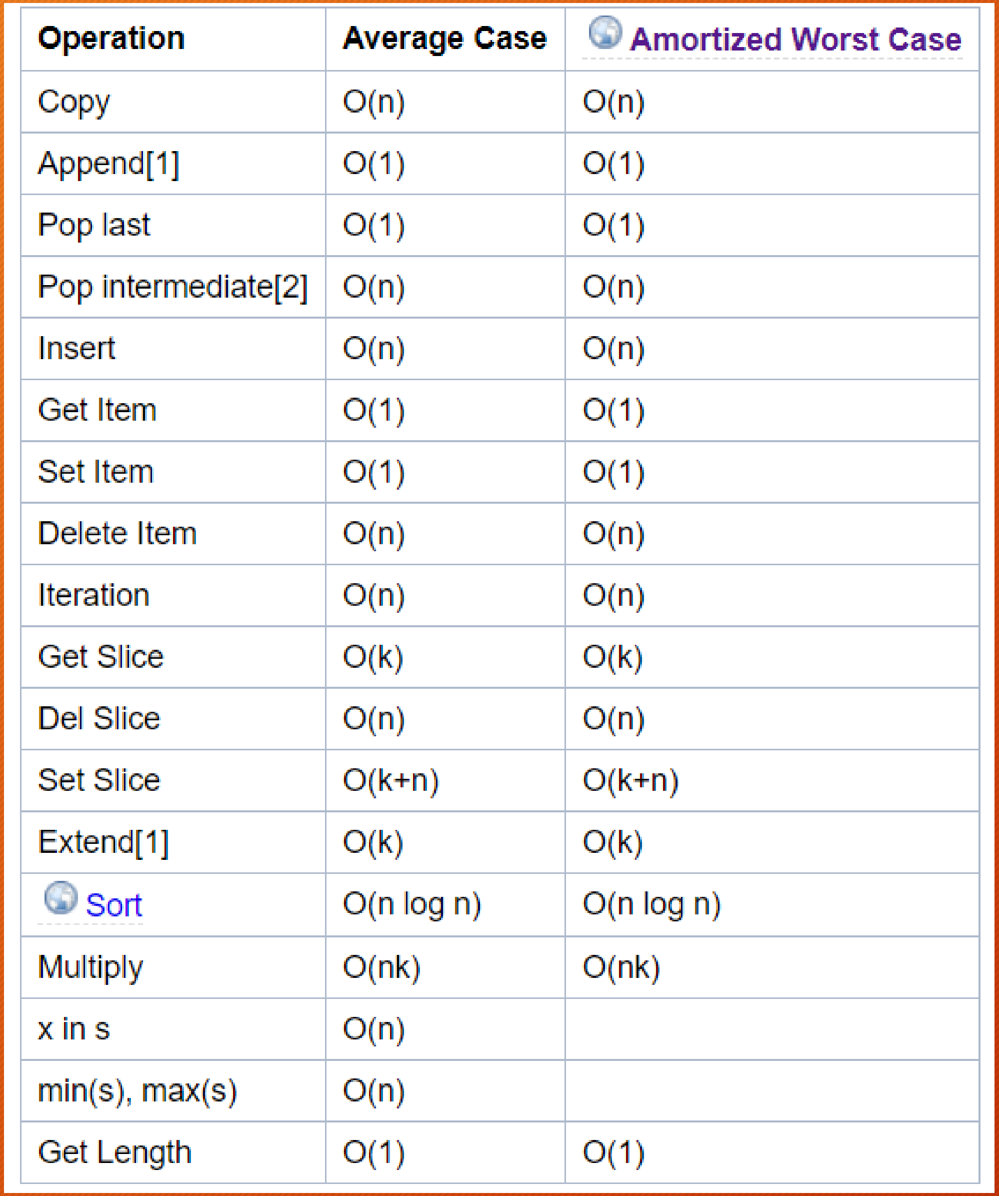
List Methods

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



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

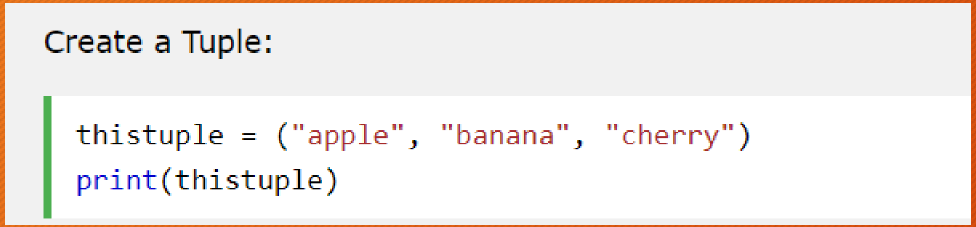
List Methods Complexity



Operation	Average Case	Amortized Worst Case
Copy	O(n)	O(n)
Append[1]	O(1)	O(1)
Pop last	O(1)	O(1)
Pop intermediate[2]	O(n)	O(n)
Insert	O(n)	O(n)
Get Item	O(1)	O(1)
Set Item	O(1)	O(1)
Delete Item	O(n)	O(n)
Iteration	O(n)	O(n)
Get Slice	O(k)	O(k)
Del Slice	O(n)	O(n)
Set Slice	O(k+n)	O(k+n)
Extend[1]	O(k)	O(k)
Sort	O(n log n)	O(n log n)
Multiply	O(nk)	O(nk)
x in s	O(n)	
min(s), max(s)	O(n)	
Get Length	O(1)	O(1)

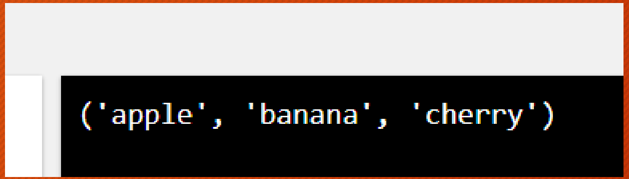
Python Tuples

Tuples are used to store multiple items in a single variable.
A tuple is a collection which is ordered and unchangeable.
Tuples are written with round brackets.



Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```



```
('apple', 'banana', 'cherry')
```

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

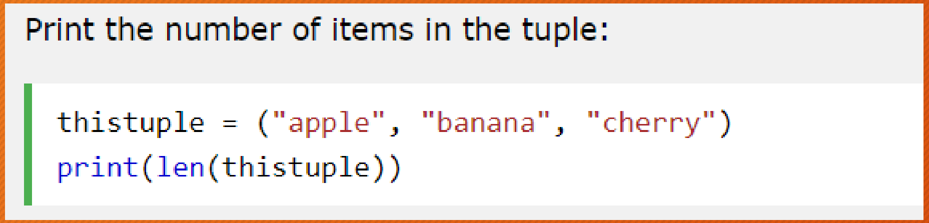
Ordered
When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable
Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates
Since tuple are indexed, tuples can have items with the same value:

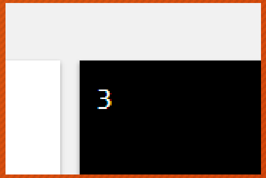
Python Tuples

To determine how many items a tuple has, use the len() function:



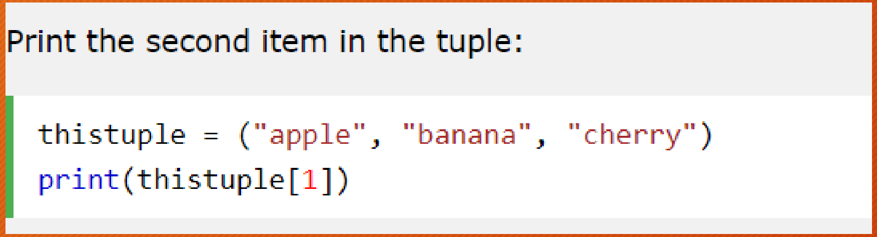
Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```



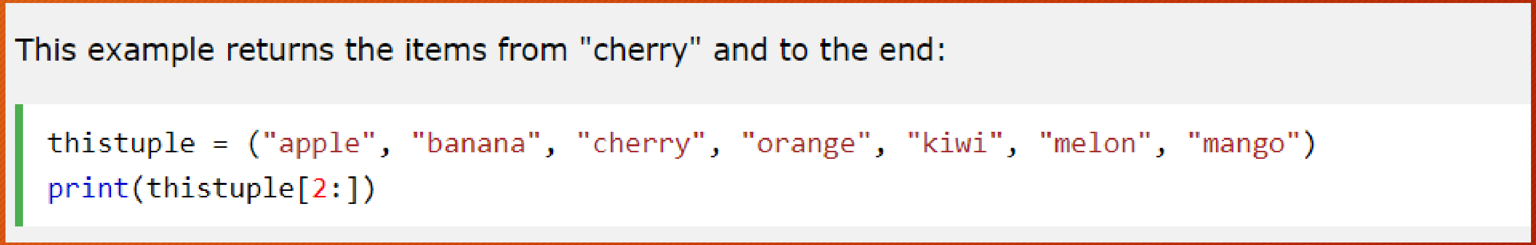
```
3
```

Access Tuple Items
You can access tuple items by referring to the index number, inside square brackets:



Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

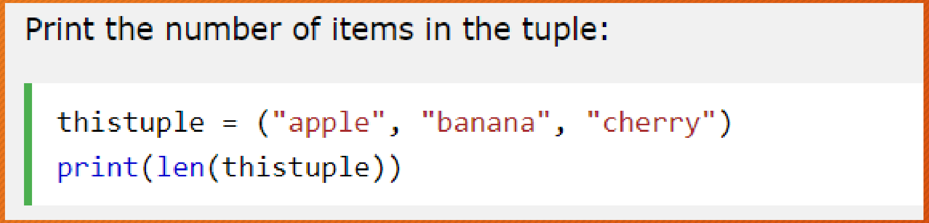


This example returns the items from "cherry" and to the end:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

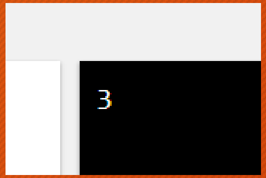
Python Tuples

To determine how many items a tuple has, use the len() function:



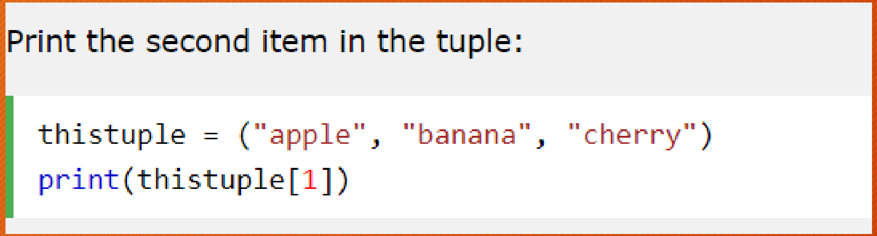
Print the number of items in the tuple:

```python
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```



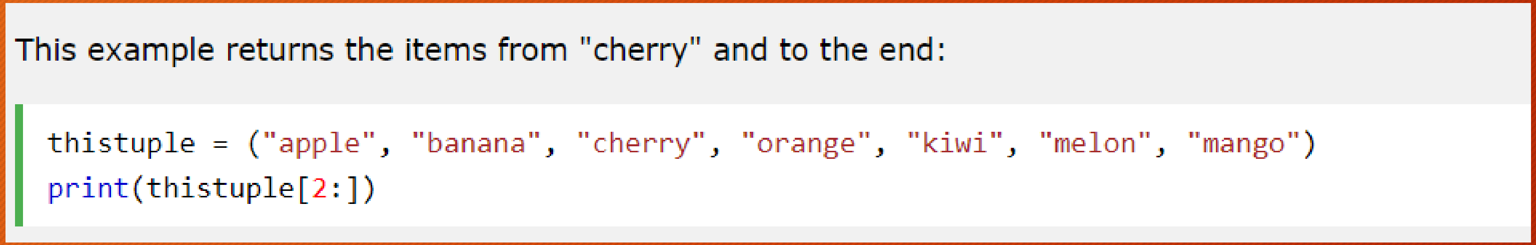
```
3
```

Access Tuple Items
You can access tuple items by referring to the index number, inside square brackets:



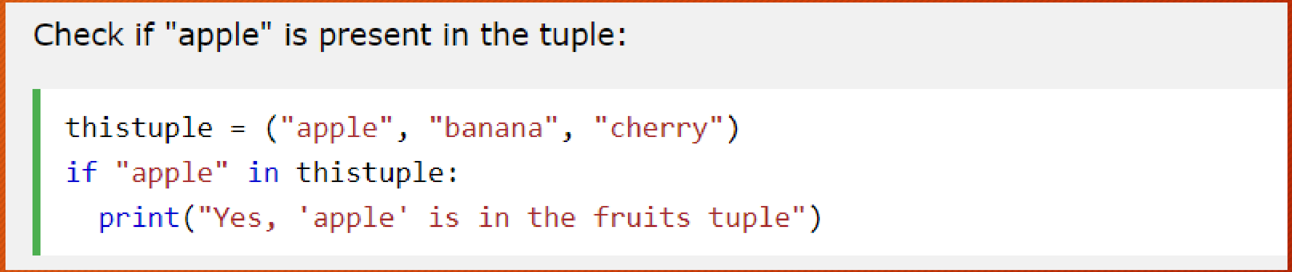
Print the second item in the tuple:

```python
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```



This example returns the items from "cherry" and to the end:

```python
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

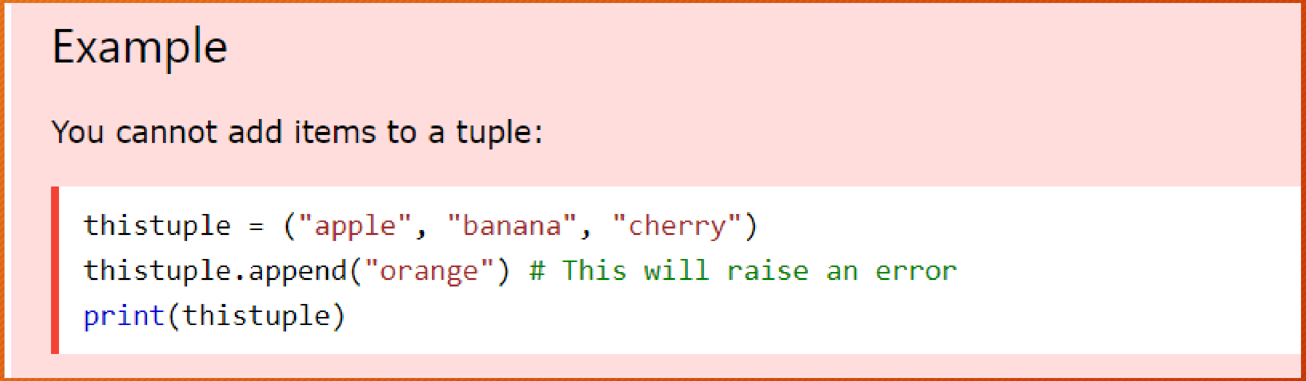


Check if "apple" is present in the tuple:

```python
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.



Example

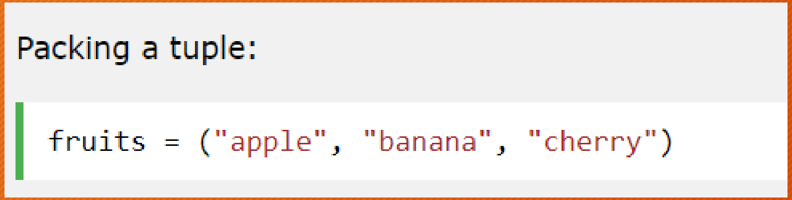
You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple.append("orange") # This will raise an error
print(thistuple)
```

Tuples are unchangeable, so you cannot remove items from it

Python - Unpack Tuples

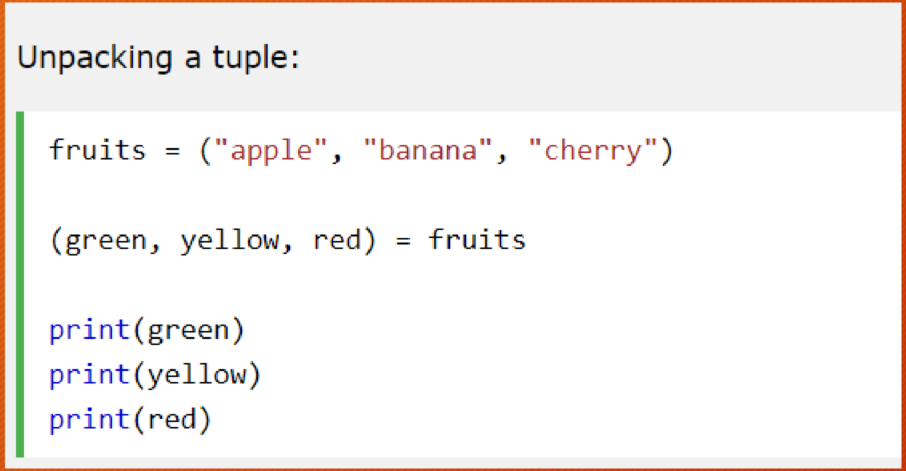
Unpacking a Tuple
When we create a tuple, we normally assign values to it. This is called "packing" a tuple:



Packing a tuple:

```python
fruits = ("apple", "banana", "cherry")
```

Unpacking a Tuple
When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

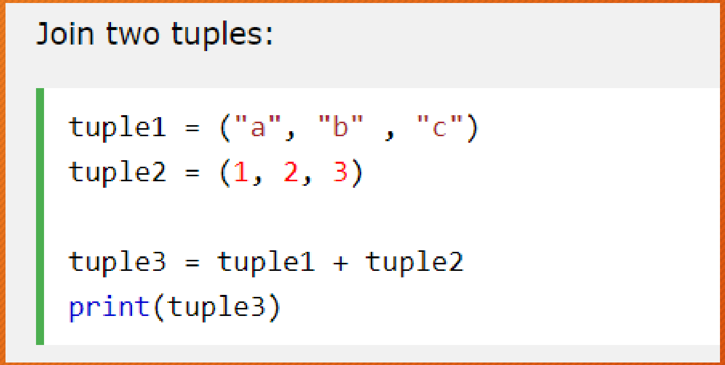


Unpacking a tuple:

```python
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Tuples

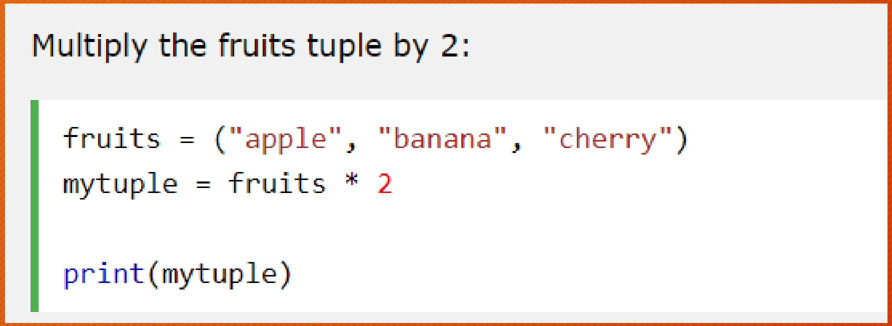
To join two or more tuples you can use the + operator



Join two tuples:

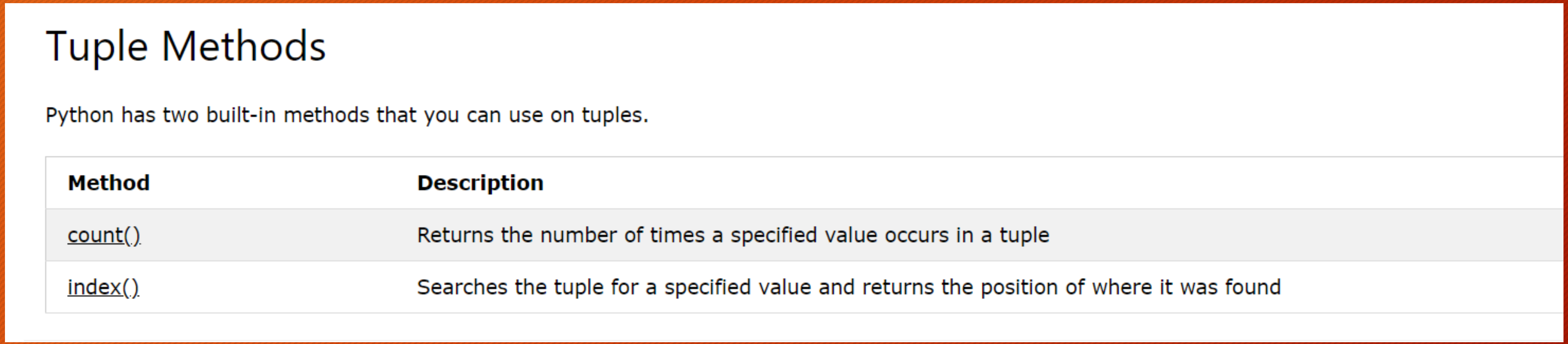
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

If you want to multiply the content of a tuple a given number of times, you can use the * operator:



Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```



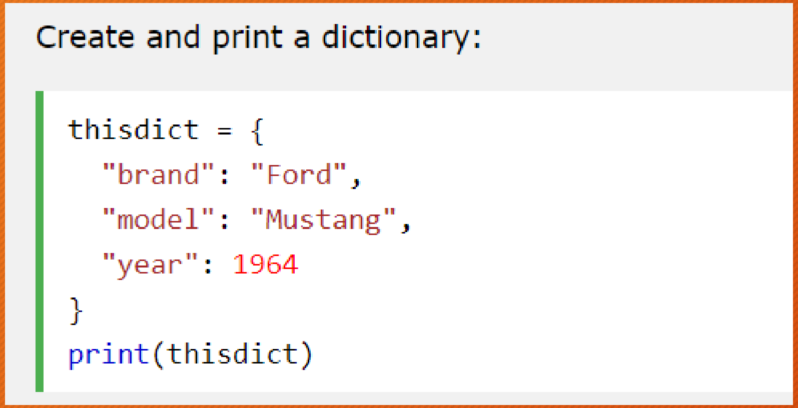
Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

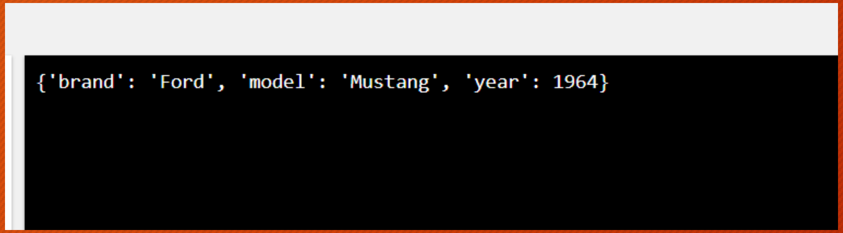
Python Dictionaries

Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is unordered, changeable and does not allow duplicates.
Dictionaries are written with curly brackets, and have keys and values:



Create and print a dictionary:

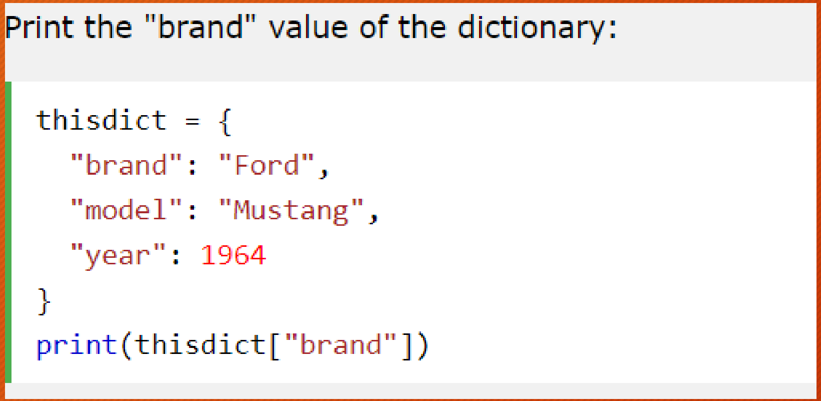
```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```



```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Dictionary Items
Dictionary items are unordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.



Print the "brand" value of the dictionary:

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Python Dictionaries

Unordered

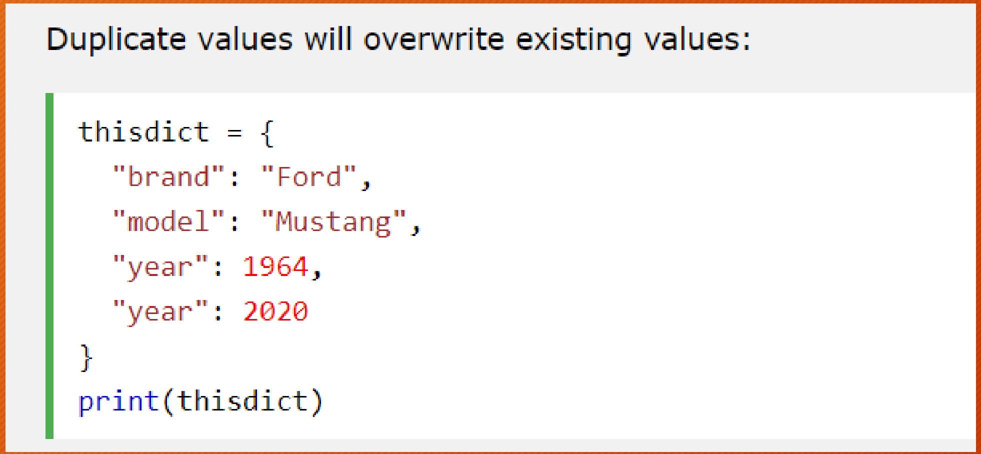
When we say that dictionaries are unordered, it means that the items does not have a defined order, you cannot refer to an item by using an index.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

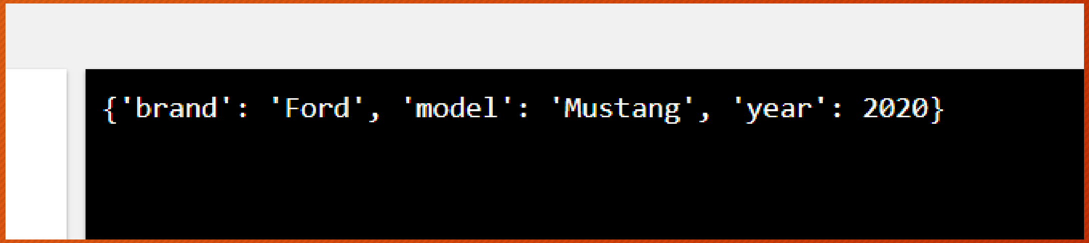
Duplicates Not Allowed

Dictionaries cannot have two items with the same key:



Duplicate values will overwrite existing values:

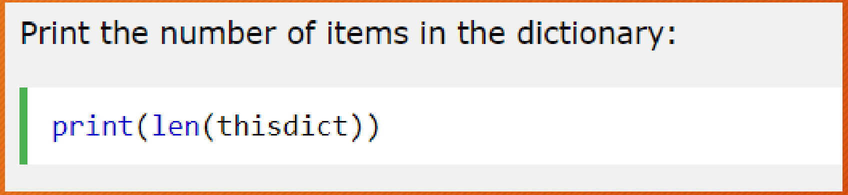
```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```



```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Python Dictionaries

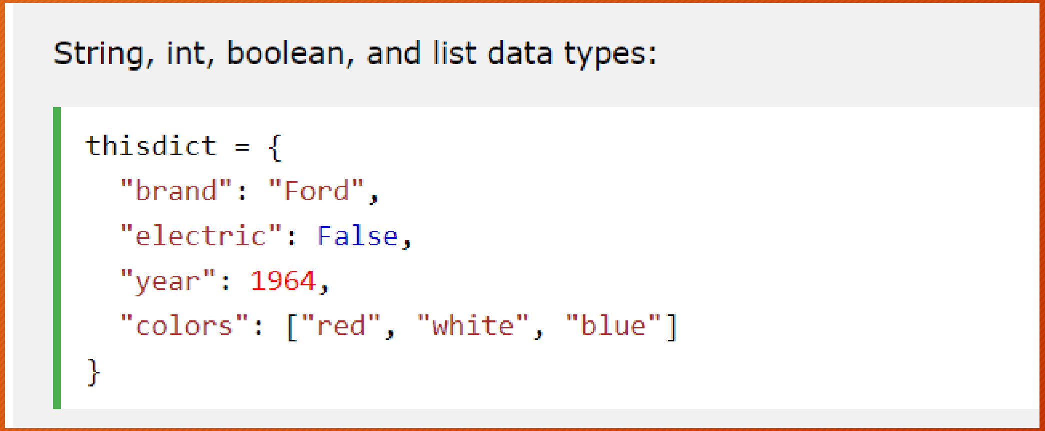
Dictionary Length
To determine how many items a dictionary has, use the len() function:



Print the number of items in the dictionary:

```python
print(len(thisdict))
```

Dictionary Length
To determine how many items a dictionary has, use the len() function:

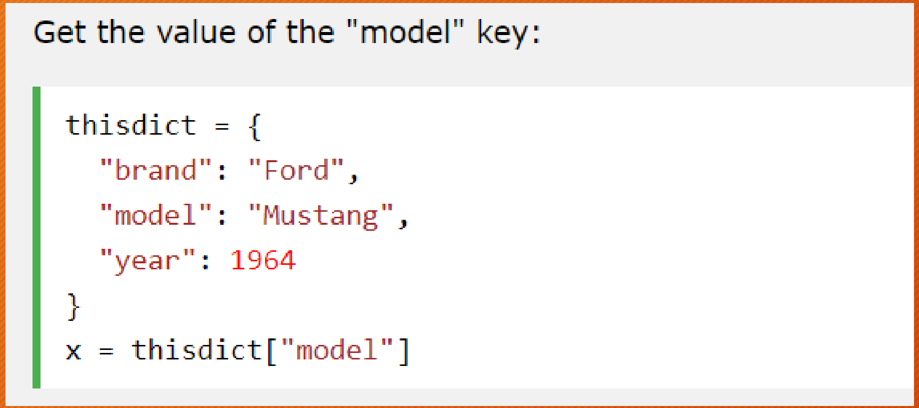


String, int, boolean, and list data types:

```python
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

Access Dictionary Items

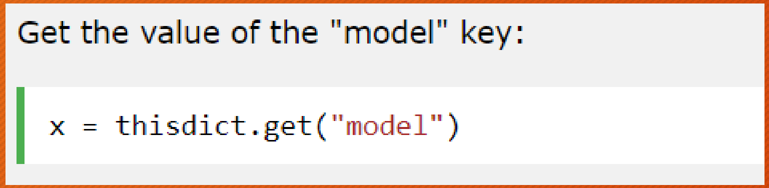
You can access the items of a dictionary by referring to its key name, inside square brackets:



Get the value of the "model" key:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

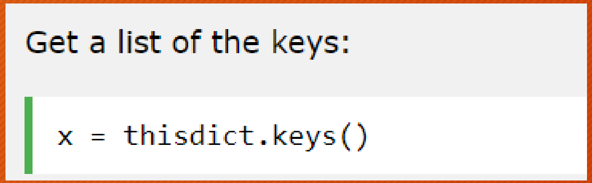
You can access the items of a dictionary by referring to its key name, inside square brackets:



Get the value of the "model" key:

```
x = thisdict.get("model")
```

The keys() method will return a list of all the keys in the dictionary.

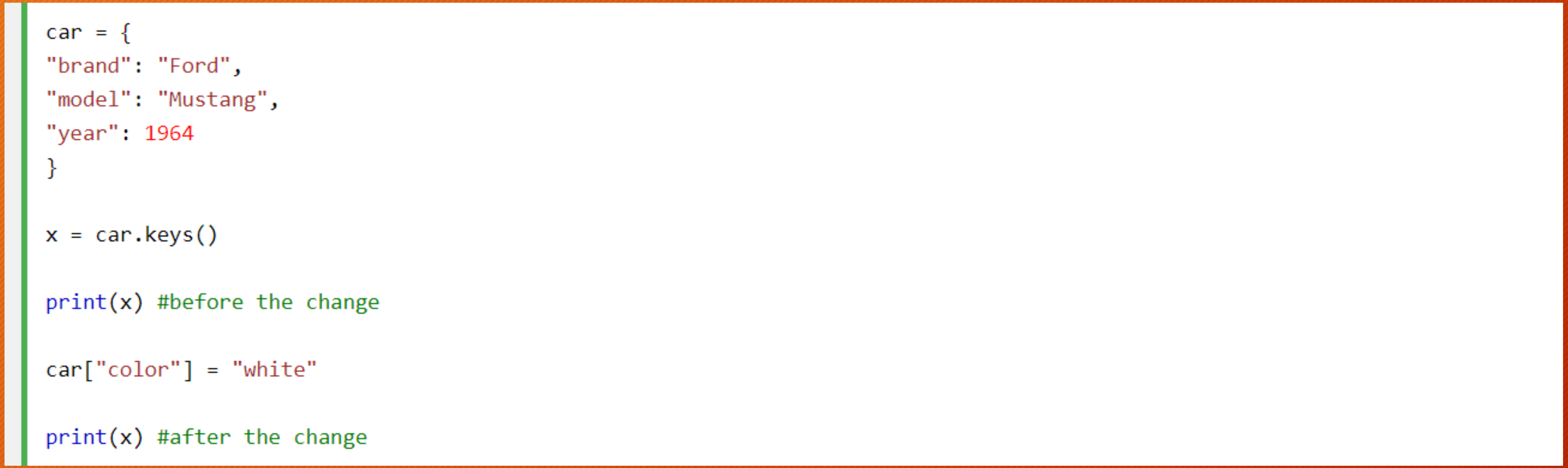


Get a list of the keys:

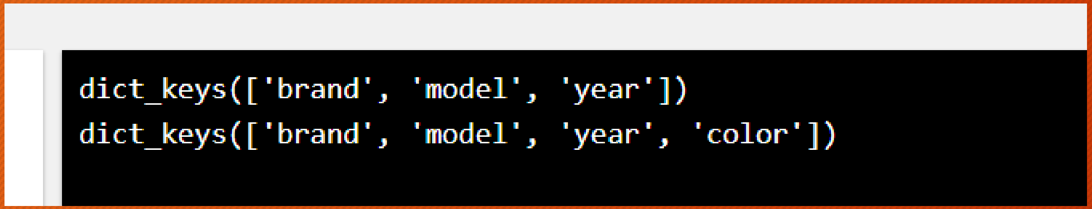
```
x = thisdict.keys()
```

Access Dictionary Items

The list of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.



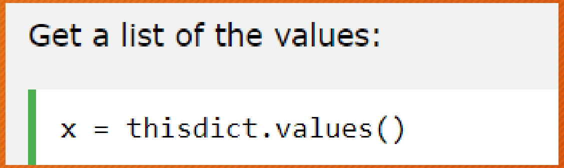
```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```



```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
```

Access Dictionary Items

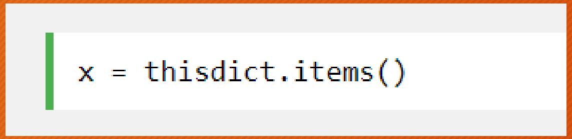
Get Values
The values() method will return a list of all the values in the dictionary.



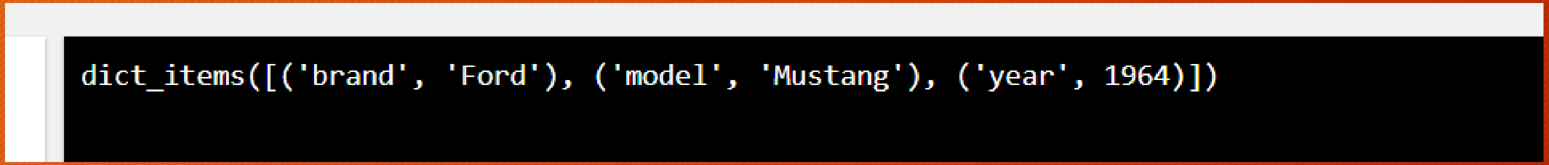
Get a list of the values:

```
x = thisdict.values()
```

The items() method will return each item in a dictionary, as tuples in a list.



```
x = thisdict.items()
```

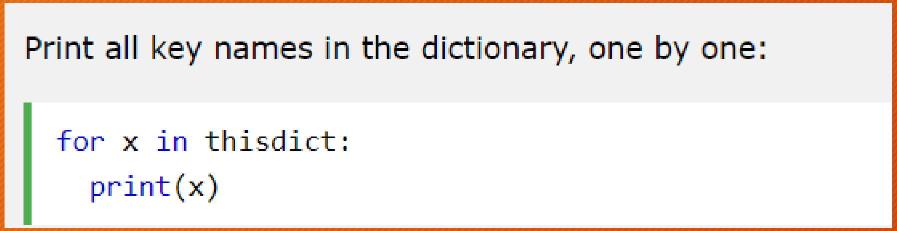


```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

Loop Through a Dictionary

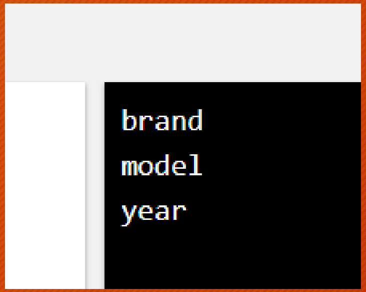
You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

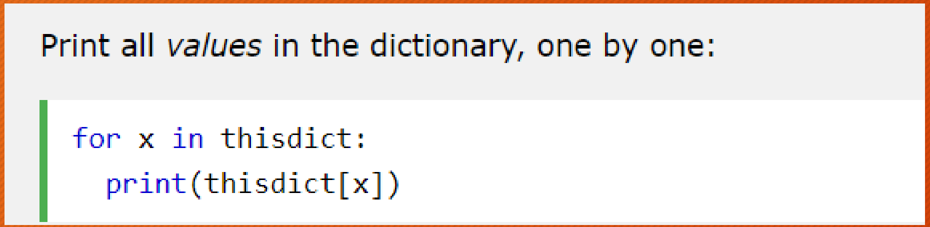


Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
```

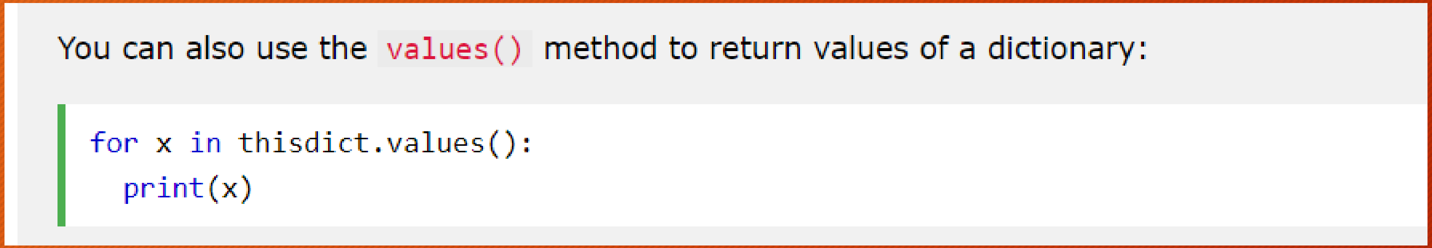


```
brand
model
year
```



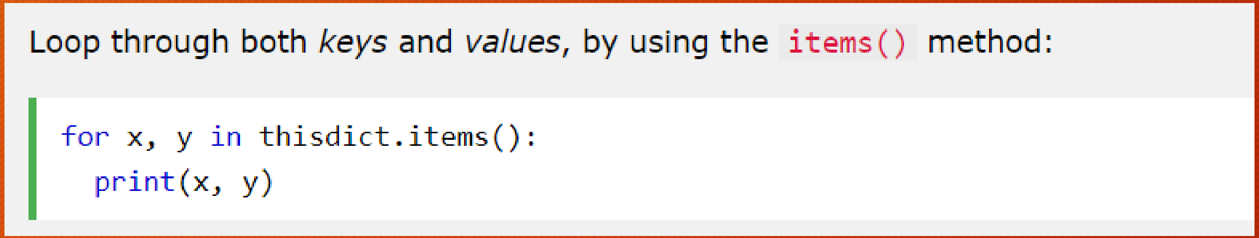
Print all *values* in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```



You can also use the `values()` method to return values of a dictionary:

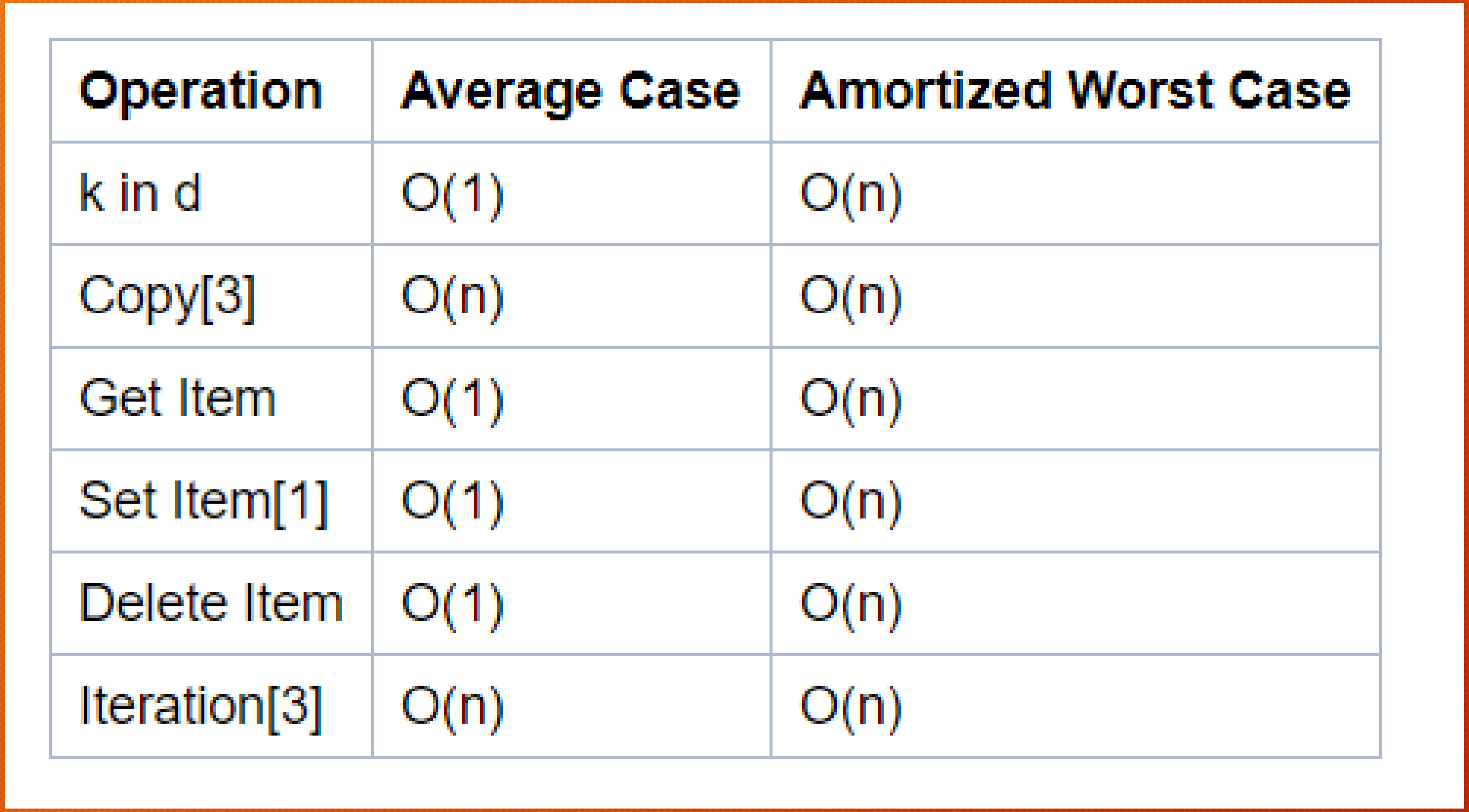
```
for x in thisdict.values():
  print(x)
```



Loop through both *keys* and *values*, by using the `items()` method:

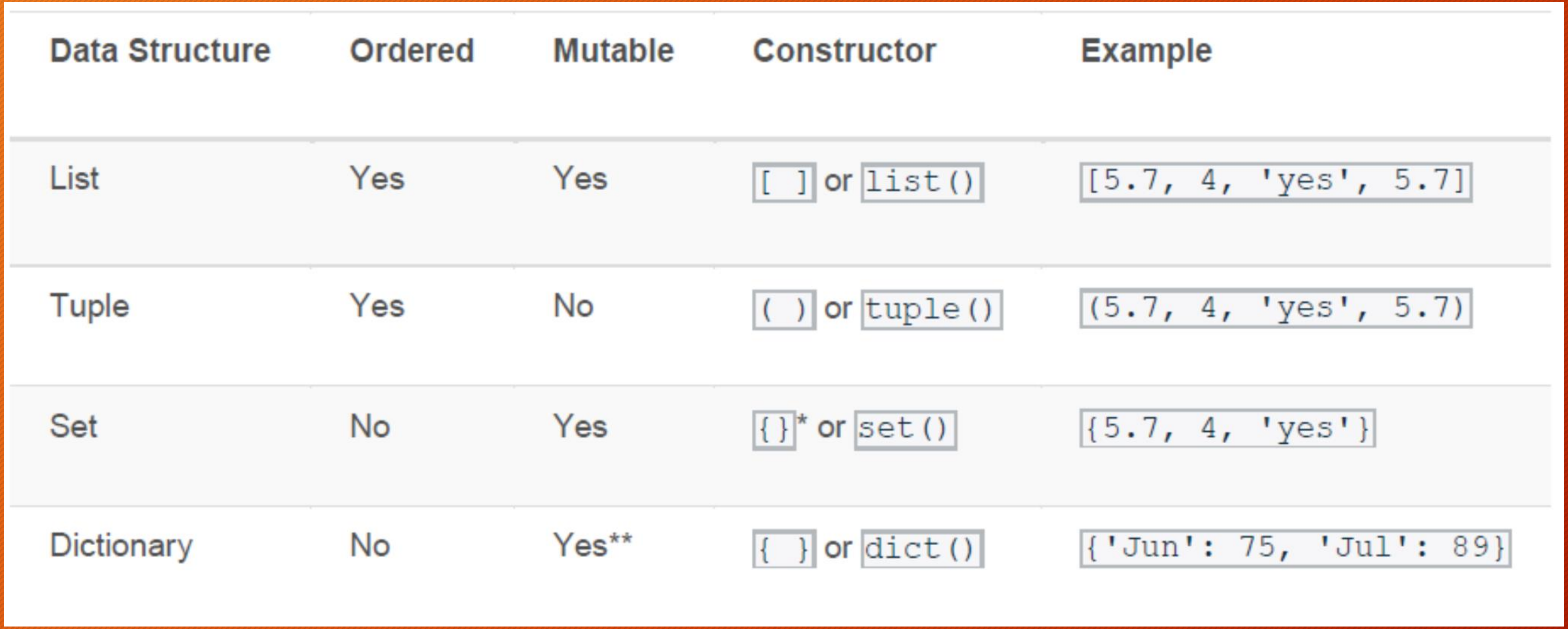
```
for x, y in thisdict.items():
  print(x, y)
```

Dictionary Methods Complexity



Operation	Average Case	Amortized Worst Case
k in d	O(1)	O(n)
Copy[3]	O(n)	O(n)
Get Item	O(1)	O(n)
Set Item[1]	O(1)	O(n)
Delete Item	O(1)	O(n)
Iteration[3]	O(n)	O(n)

python data structures



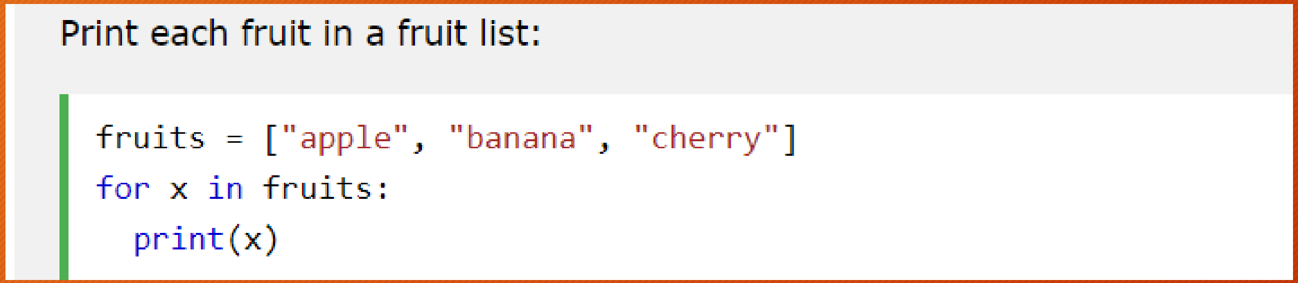
Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Python For Loops

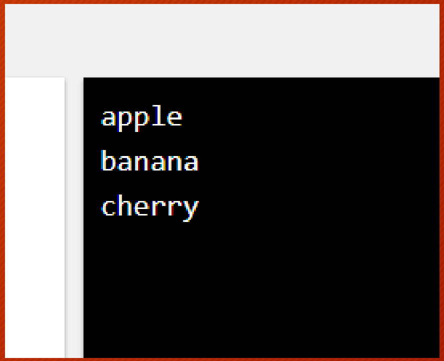
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

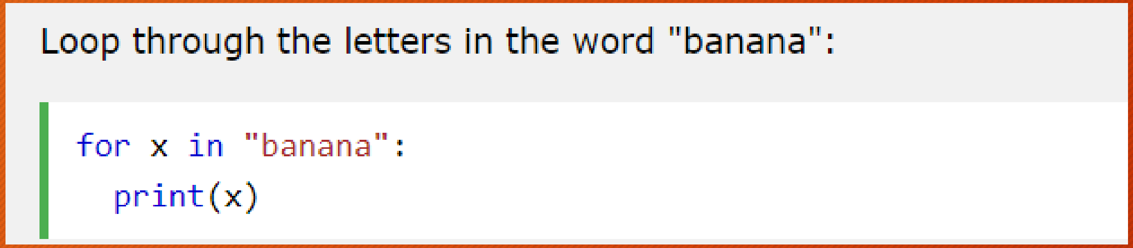
With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.


Print each fruit in a fruit list:

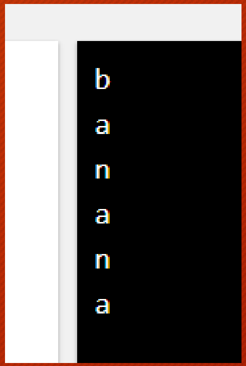
```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```



```
apple
banana
cherry
```


Loop through the letters in the word "banana":

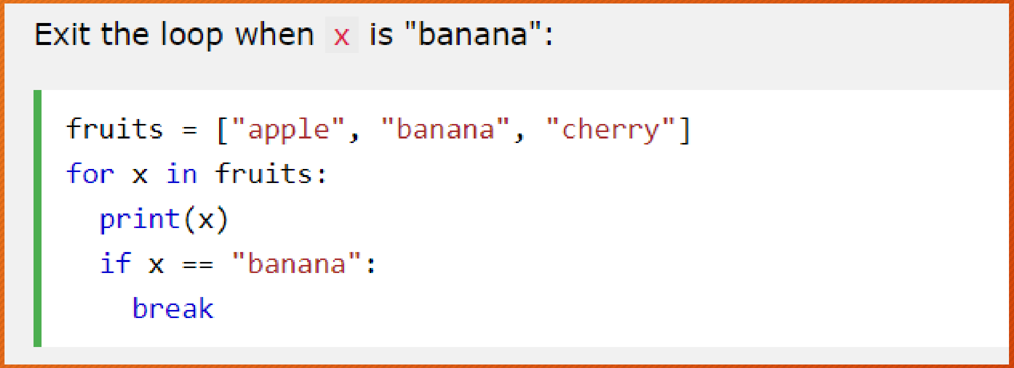
```python
for x in "banana":
  print(x)
```



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

Python For Loops

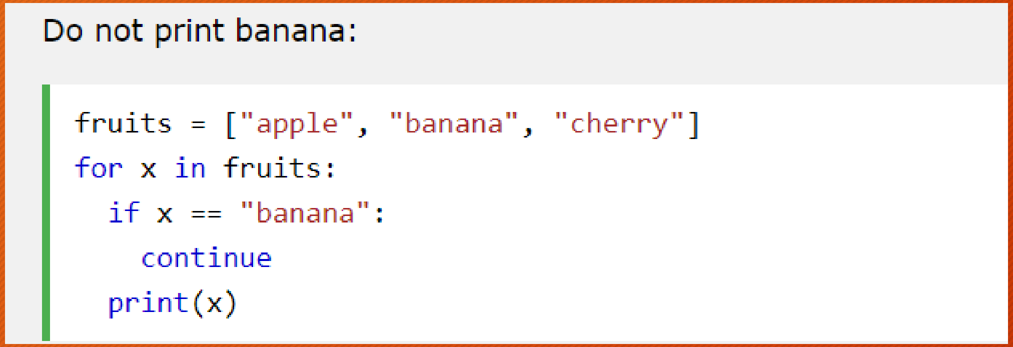
With the break statement we can stop the loop before it has looped through all the items:



Exit the loop when x is "banana":

```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

With the continue statement we can stop the current iteration of the loop, and continue with the next:

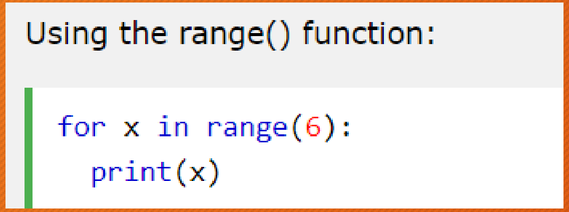


Do not print banana:

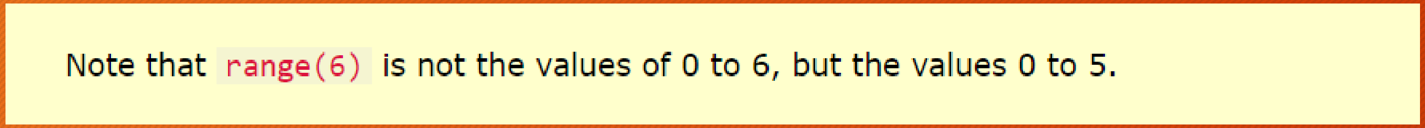
```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

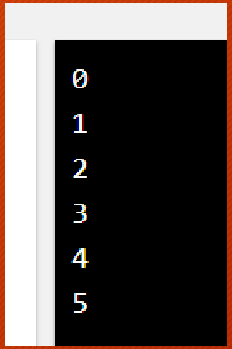
Python For Loops The range() Function

To loop through a set of code a specified number of times, we can use the range() function,
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.


Using the range() function:

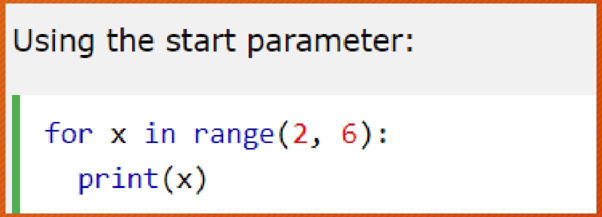
```
for x in range(6):
  print(x)
```


Note that `range(6)` is not the values of 0 to 6, but the values 0 to 5.

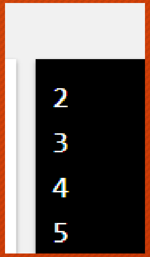


```
0
1
2
3
4
5
```

To loop through a set of code a specified number of times, we can use the range() function,
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

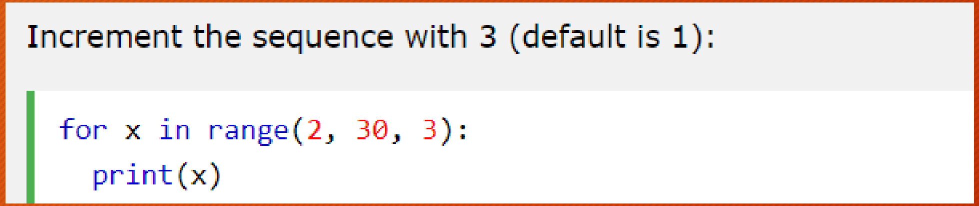

Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

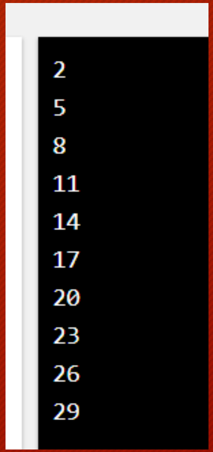


```
2
3
4
5
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):


Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

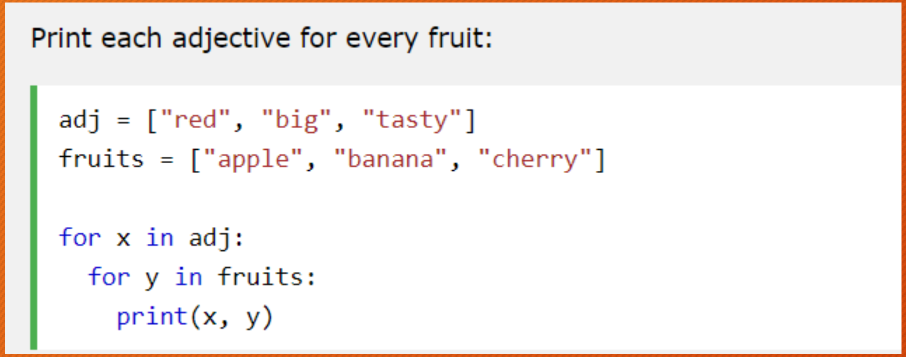


```
2
5
8
11
14
17
20
23
26
29
```

Python For Loops

A nested loop is a loop inside a loop.

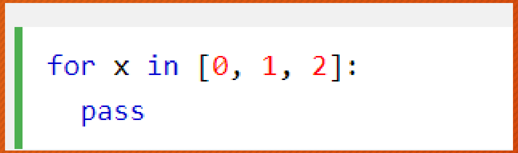
The "inner loop" will be executed one time for each iteration of the "outer loop":



Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

The pass Statement
for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.



```
for x in [0, 1, 2]:
  pass
```

Python Functions

In Python a function is defined using the **def** keyword:

Example

```
def my_function():
  print("Hello from a function")
```

```
my_function()
```

To call a function, use the function name followed by parenthesis:

Information can be passed into functions as arguments.

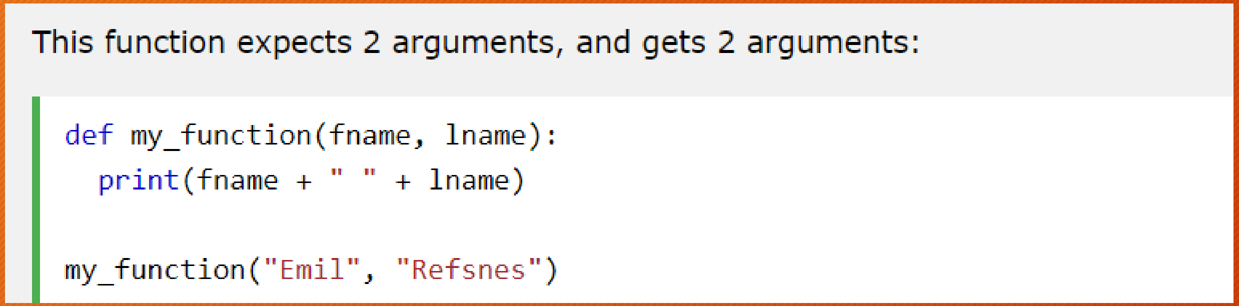
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Python Functions

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

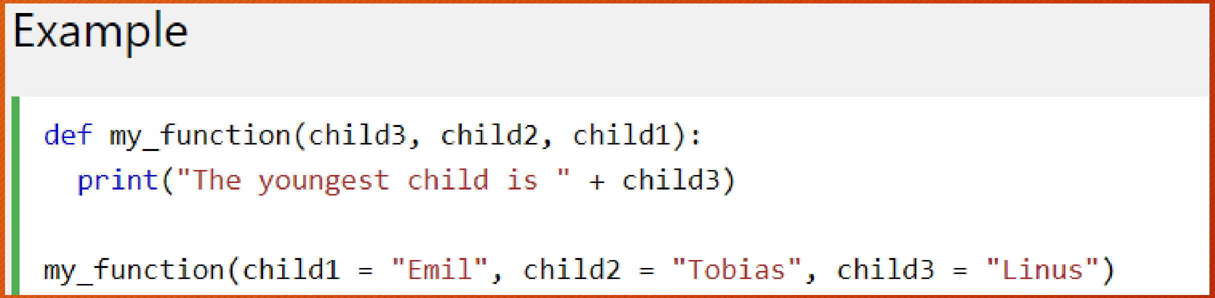


This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

Keyword Arguments
You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.



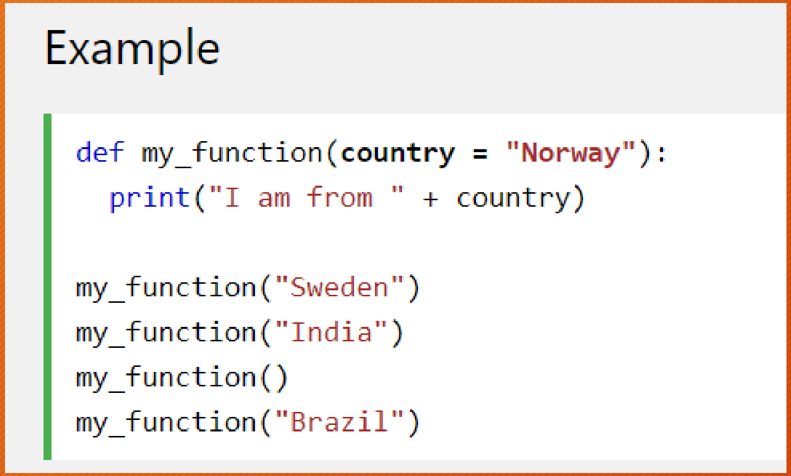
Example

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Python Functions

Default Parameter Value
The following example shows how to use a default parameter value.

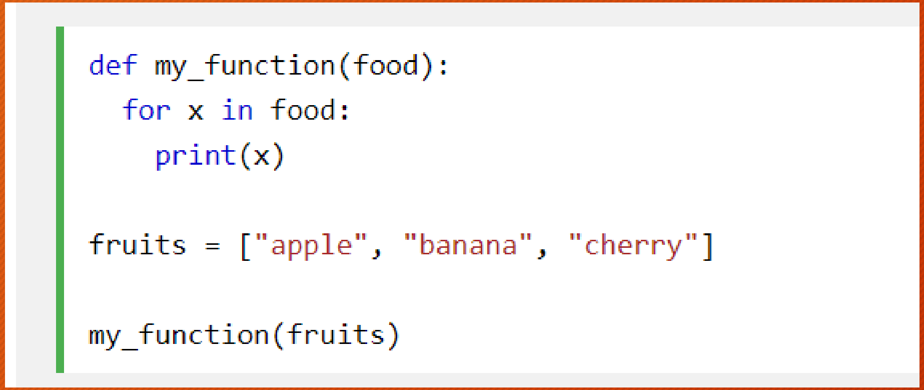
If we call the function without argument, it uses the default value:



Example

```python
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

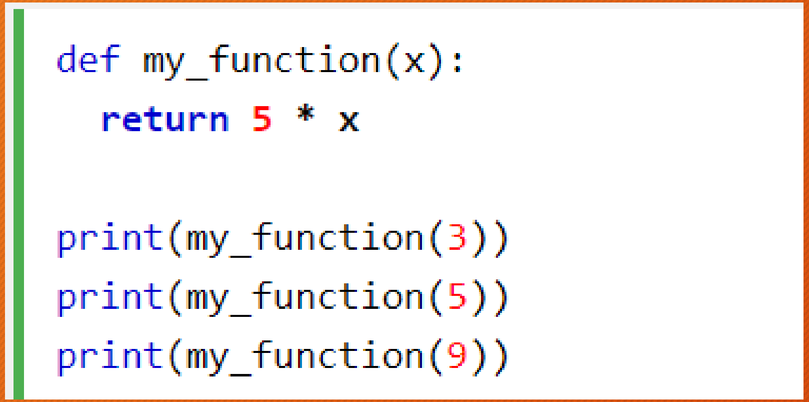
You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
E.g. if you send a List as an argument, it will still be a List when it reaches the function:



```python
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

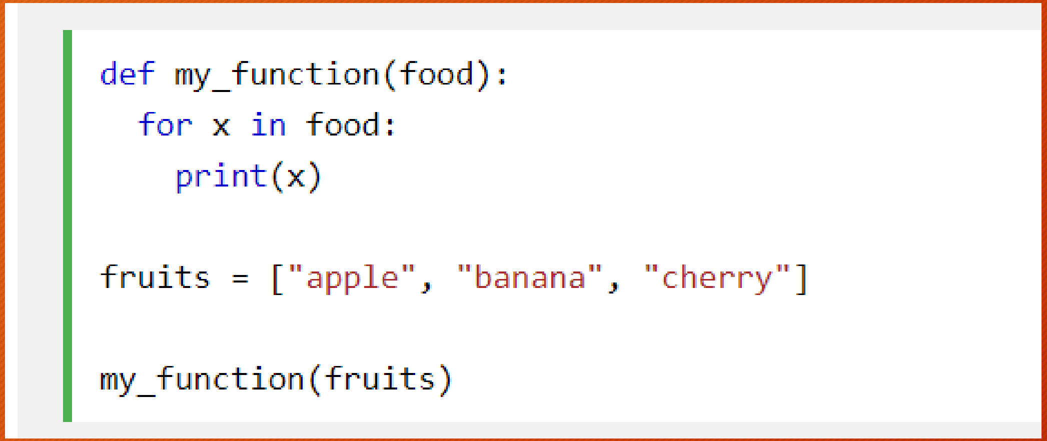
Python Functions

Return Values
To let a function return a value, use the return statement:



```python
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

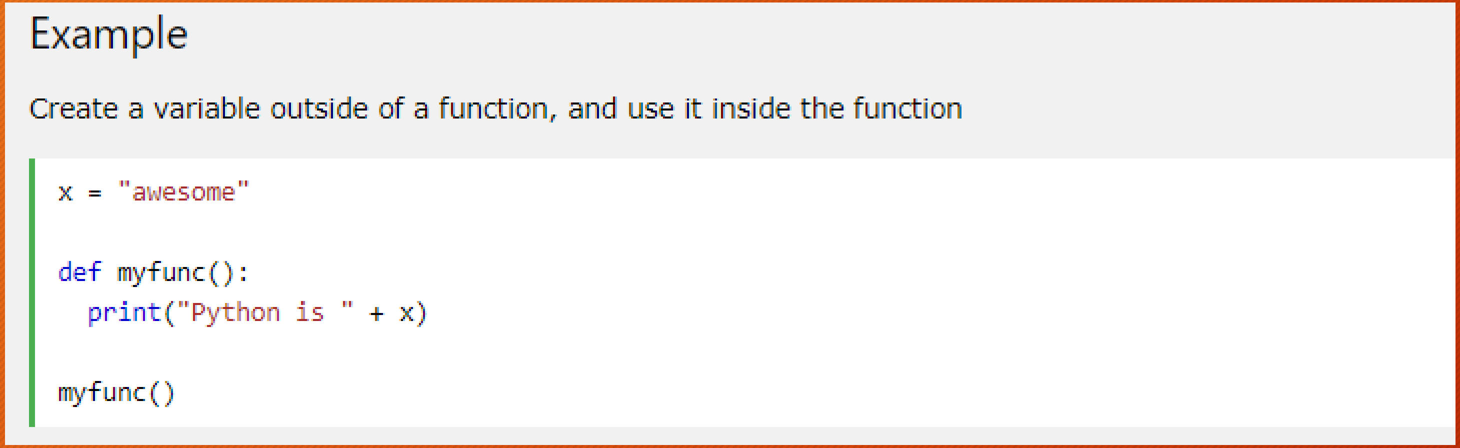
The pass Statement
function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.



```python
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Python - Global Variables

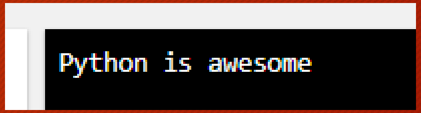
Global Variables
Variables that are created outside of a function (as in all of the examples above) are known as global variables.
Global variables can be used by everyone, both inside of functions and outside.



Example

Create a variable outside of a function, and use it inside the function

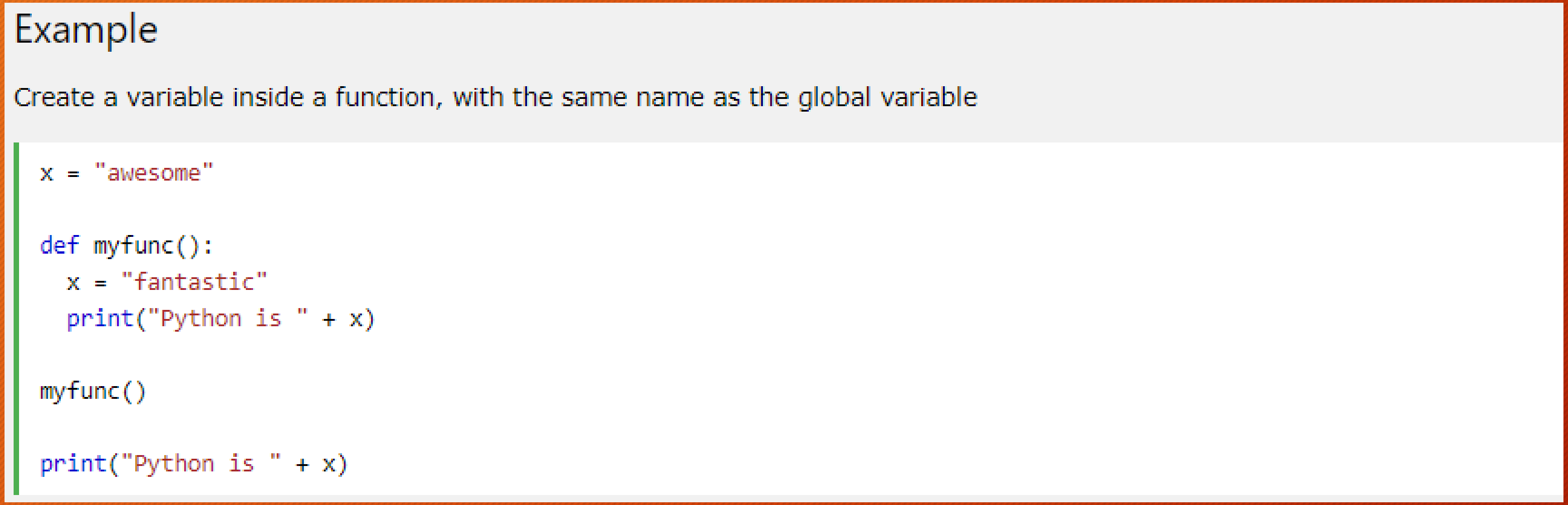
```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```



```
Python is awesome
```

Python - Global Variables

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.



Example

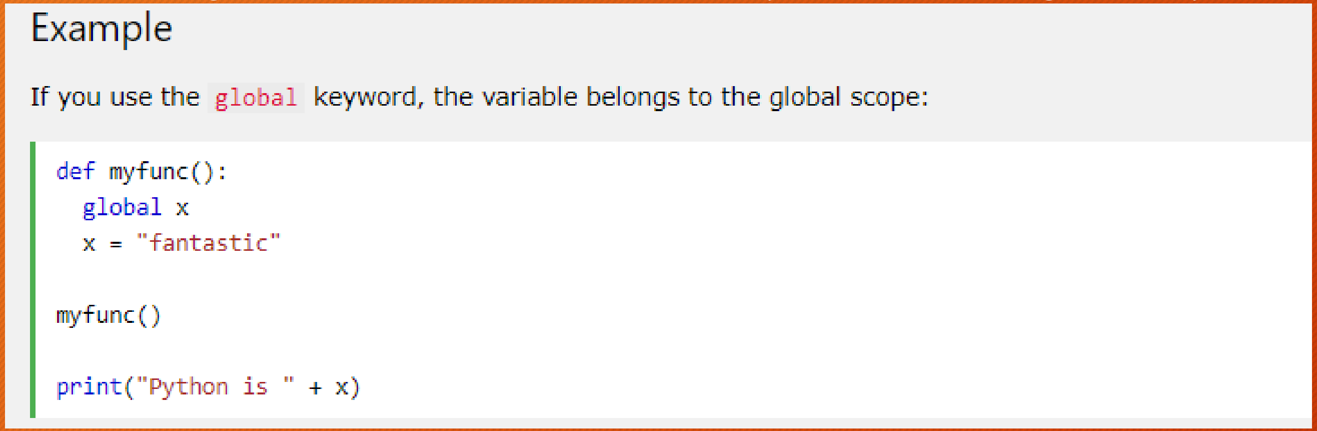
Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

Python - Global Variables

The global Keyword
Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

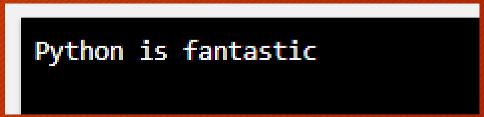
To create a global variable inside a function, you can use the global keyword.



Example

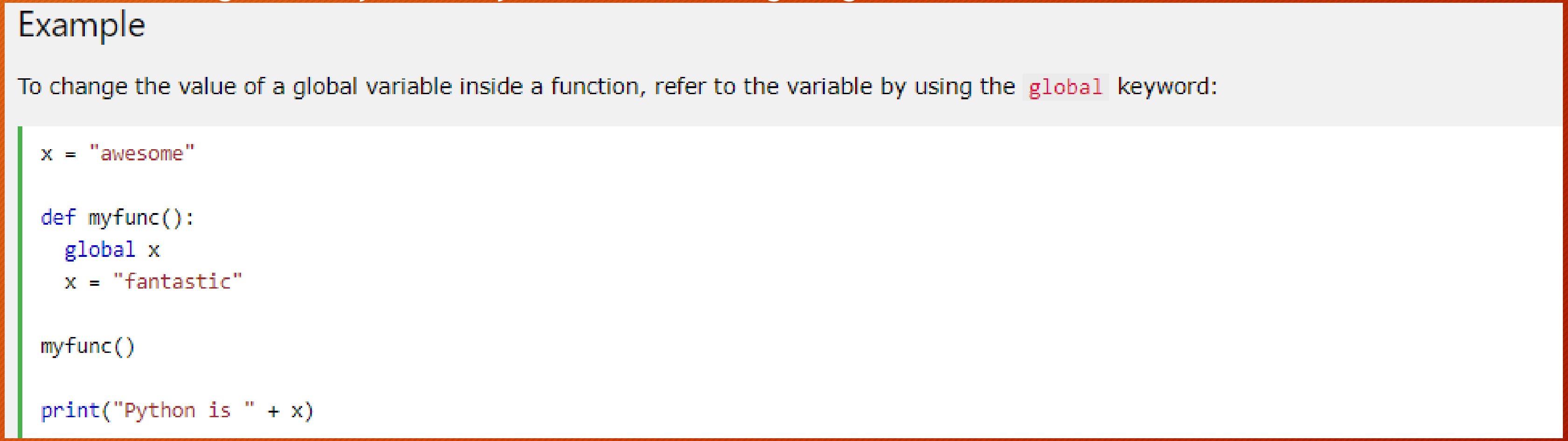
If you use the `global` keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```



```
Python is fantastic
```

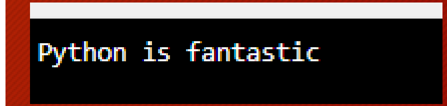
Also, use the global keyword if you want to change a global variable inside a function.



Example

To change the value of a global variable inside a function, refer to the variable by using the `global` keyword:

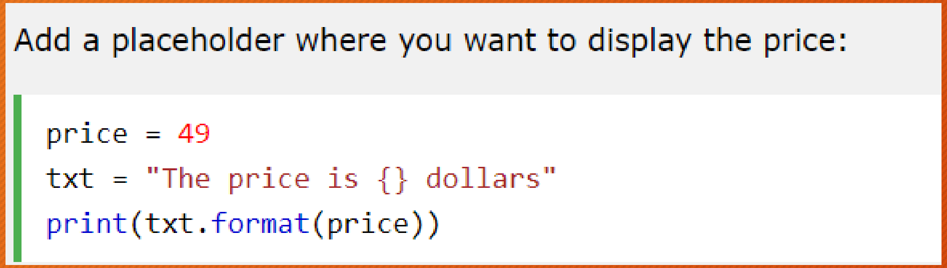
```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```



```
Python is fantastic
```

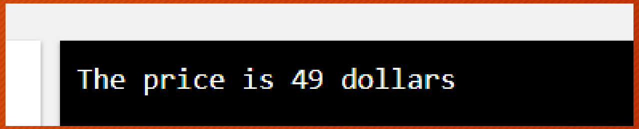
Python String Formatting

To make sure a string will display as expected, we can format the result with the format() method.
The format() method allows you to format selected parts of a string.
Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?
To control such values, add placeholders (curly brackets {}) in the text, and run the values through the format() method:



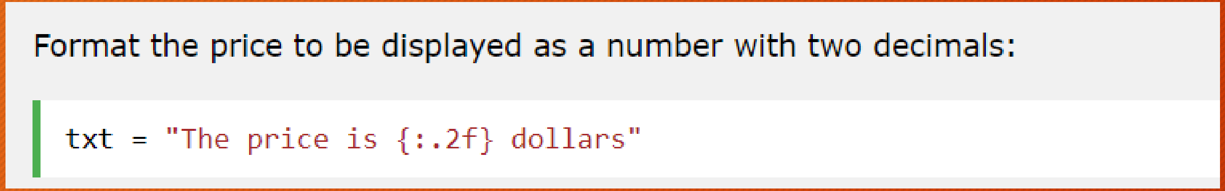
Add a placeholder where you want to display the price:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```



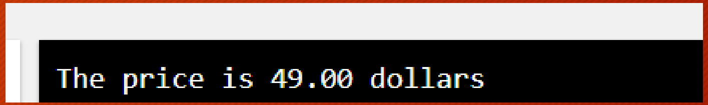
```
The price is 49 dollars
```

You can add parameters inside the curly brackets to specify how to convert the value:



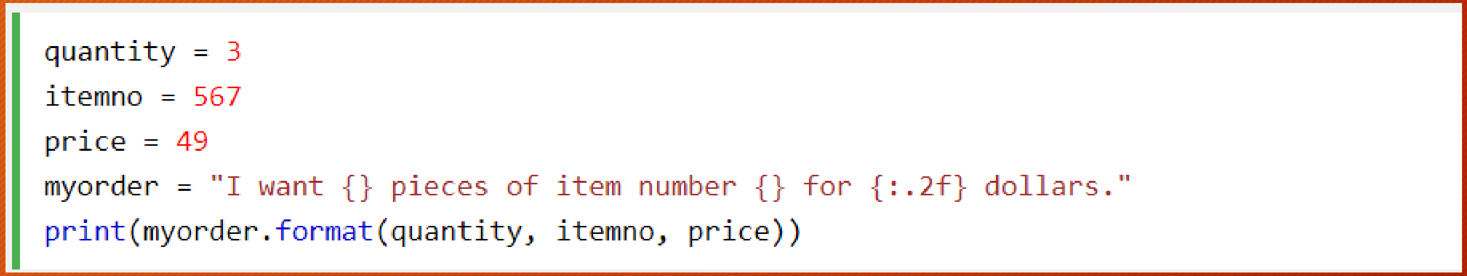
Format the price to be displayed as a number with two decimals:

```
txt = "The price is {:.2f} dollars"
```

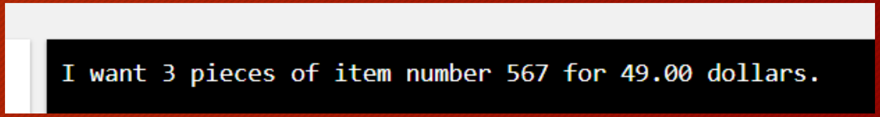


```
The price is 49.00 dollars
```

If you want to use more values, just add more values to the format() method:



```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```



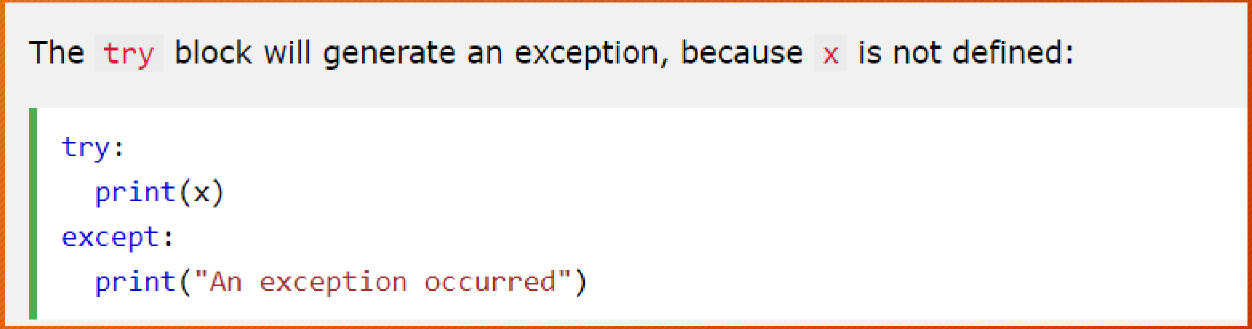
```
I want 3 pieces of item number 567 for 49.00 dollars.
```

Python Try Except

The try block lets you test a block of code for errors.

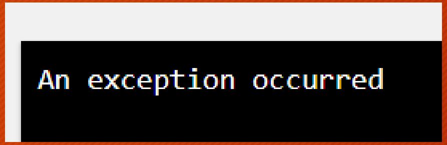
The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.



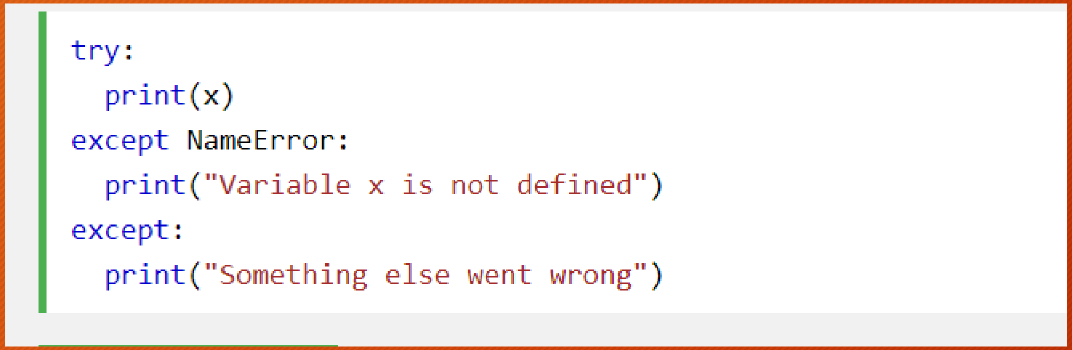
The try block will generate an exception, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```



```
An exception occurred
```

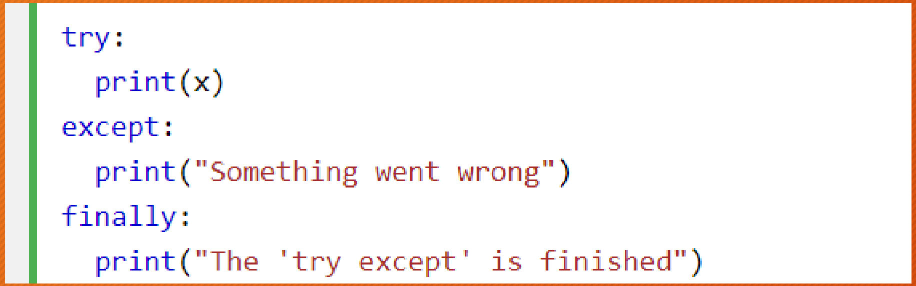
You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:



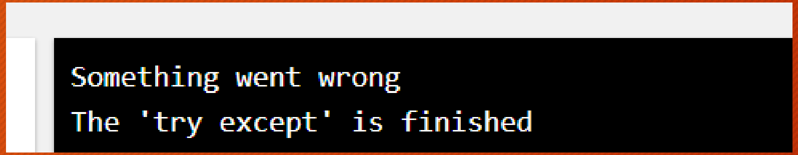
```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Python Try Except

The finally block, if specified, will be executed regardless if the try block raises an error or not.

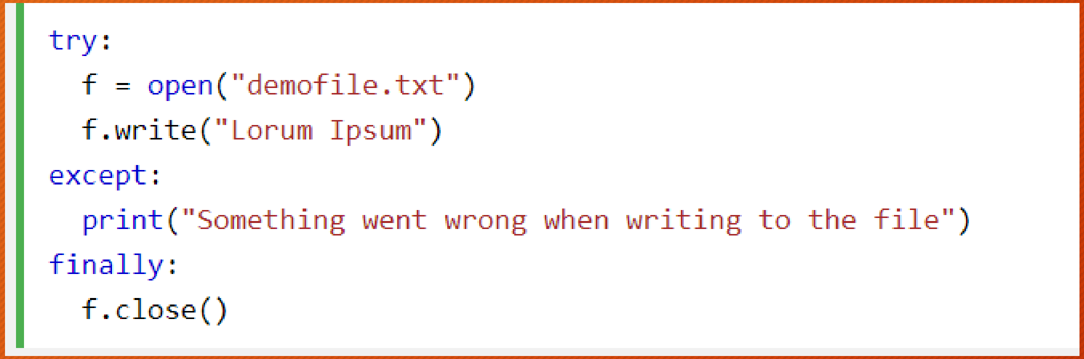


```python
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```



```
Something went wrong
The 'try except' is finished
```

This can be useful to close objects and clean up resources:

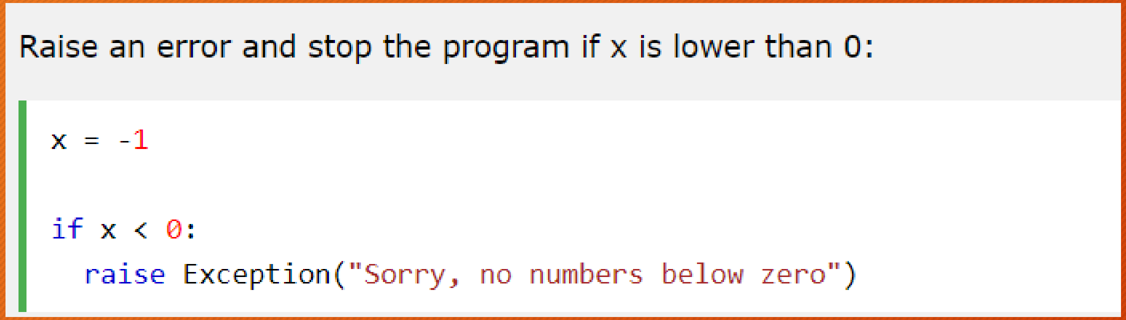


```python
try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword

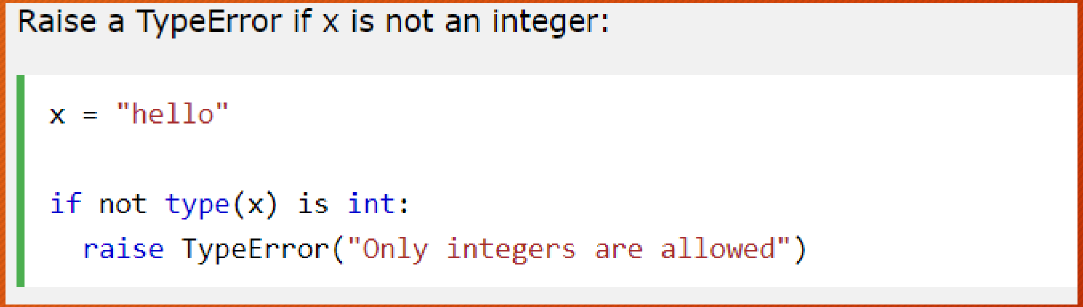


Raise an error and stop the program if x is lower than 0:

```python
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

The raise keyword is used to raise an exception.

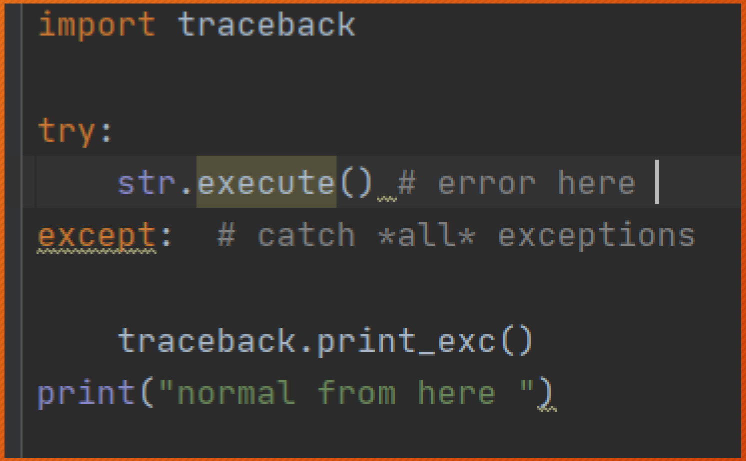
You can define what kind of error to raise, and the text to print to the user.



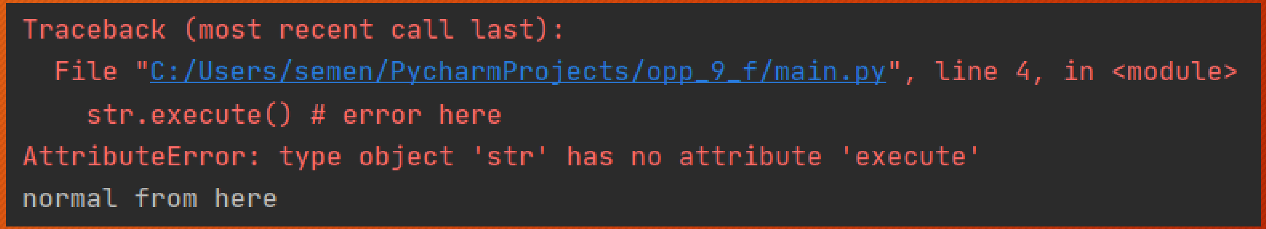
Raise a TypeError if x is not an integer:

```python
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

catch all exceptions



```python
import traceback

try:
    str.execute() # error here
except:  # catch *all* exceptions

    traceback.print_exc()
print("normal from here ")
```

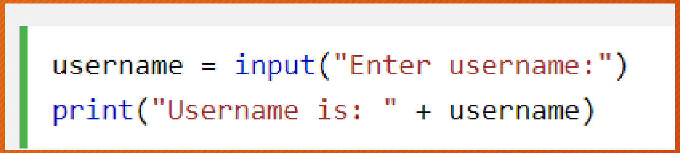


```
Traceback (most recent call last):
  File "C:/Users/semen/PycharmProjects/opp_9_f/main.py", line 4, in <module>
    str.execute() # error here
AttributeError: type object 'str' has no attribute 'execute'
normal from here
```

Notice the normal from here.

Python User Input

Python 3.6 uses the input() method.



```
username = input("Enter username:")
print("Username is: " + username)
```

Sources

https://www2.slideshare.net/LiliaSfaxi/software-engineering-chp4-design-patterns

https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples