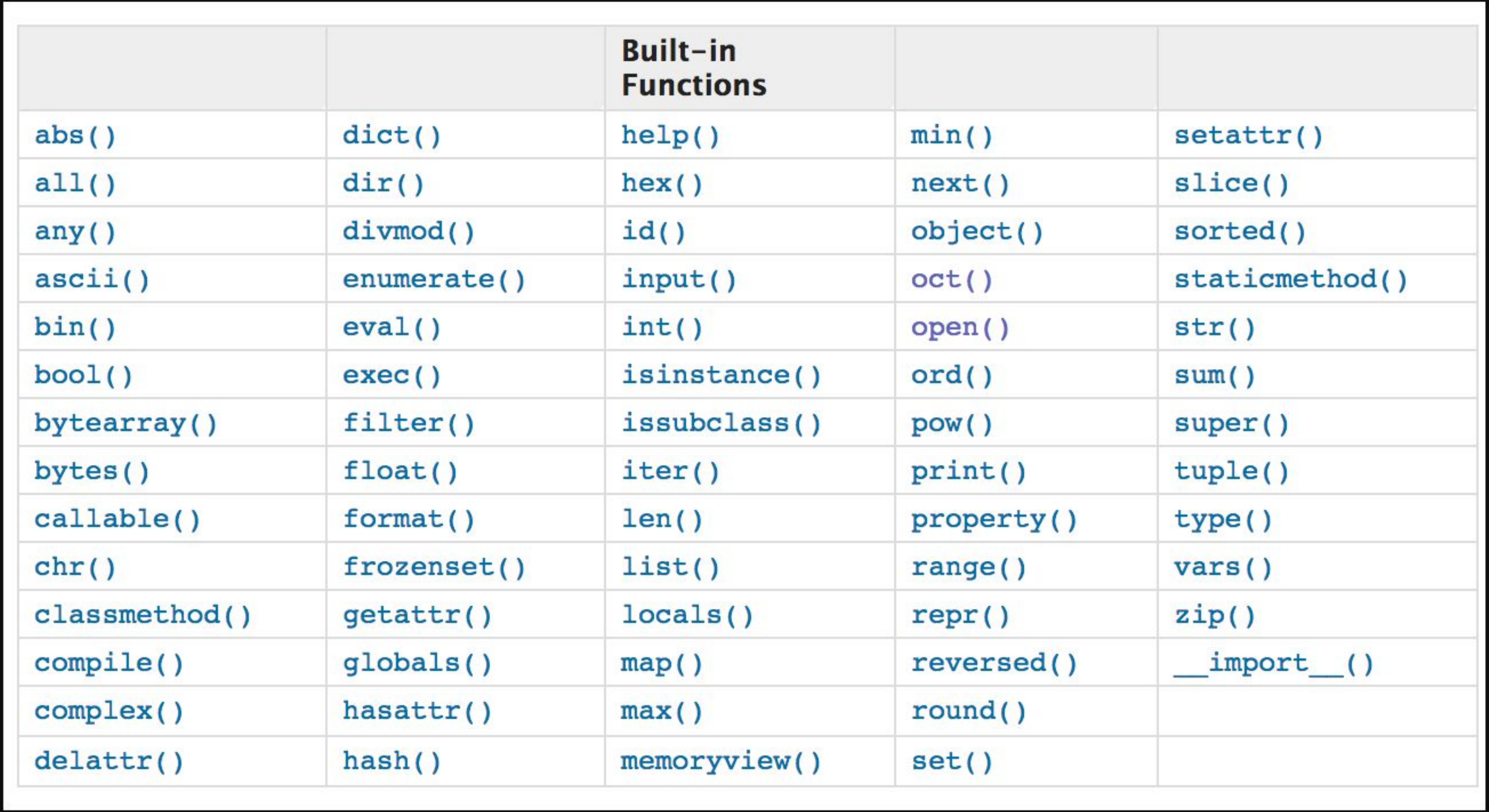
- Python Built in Functions
- Math module
- Tuple Concepts
- Tuple Built in Functions
- Examples of Tuple
- Enumerate
- Dictionary Concepts
- Dictionary Examples

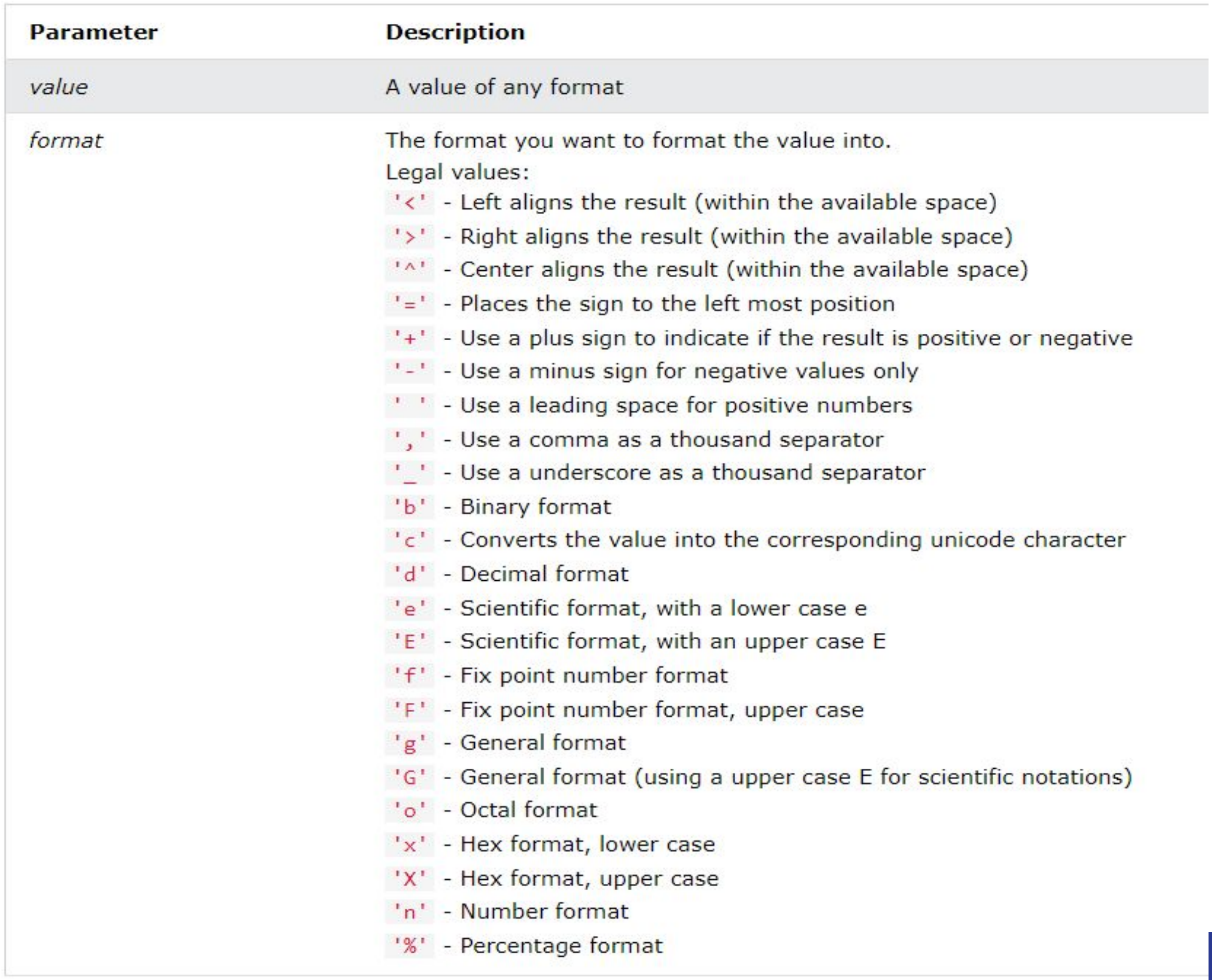
Python Built in Functions

- len() → To find the length of the given sequence.
- int (value, base) →
  - Base: A number representing the number format. Default value: 10
  - Value: A number or a string that can be converted into an integer number
- bool() → function returns the boolean value of a specified object.
- bin() → converts to binary format
- The **`chr()`** function returns the character that represents the specified unicode.
- The **`hex()`** function converts the specified number into a hexadecimal value.
- The **`oct()`** function converts an integer into an octal string.
- **abs()**



		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Parameter Values



Parameter	**Description**
value	A value of any format
format	The format you want to format the value into. Legal values:
	`'<'` - Left aligns the result (within the available space)
	`'>'` - Right aligns the result (within the available space)
	`'^'` - Center aligns the result (within the available space)
	`'='` - Places the sign to the left most position
	`'+'` - Use a plus sign to indicate if the result is positive or negative
	`'-'` - Use a minus sign for negative values only
	`' '` - Use a leading space for positive numbers
	`','` - Use a comma as a thousand separator
	`'_'` - Use a underscore as a thousand separator
	`'b'` - Binary format
	`'c'` - Converts the value into the corresponding unicode character
	`'d'` - Decimal format
	`'e'` - Scientific format, with a lower case e
	`'E'` - Scientific format, with an upper case E
	`'f'` - Fix point number format
	`'F'` - Fix point number format, upper case
	`'g'` - General format
	`'G'` - General format (using a upper case E for scientific notations)
	`'o'` - Octal format
	`'x'` - Hex format, lower case
	`'X'` - Hex format, upper case
	`'n'` - Number format
	`'%'` - Percentage format

Math Module → import math

- math.ceil() = 4.2 → 5
- math.floor() = 3.6 → 3
- math.factorial() = 5 → 120
- math.fabs() = -3.45 → 3.45
- math.gcd() = 3, 6 → 3
- math.pow(4, 3) = 64.0
- math.pi = 3.142
- math.sqrt(9) = 3
- math.lcm(2, 4, 6) = 12

Tuple in Python

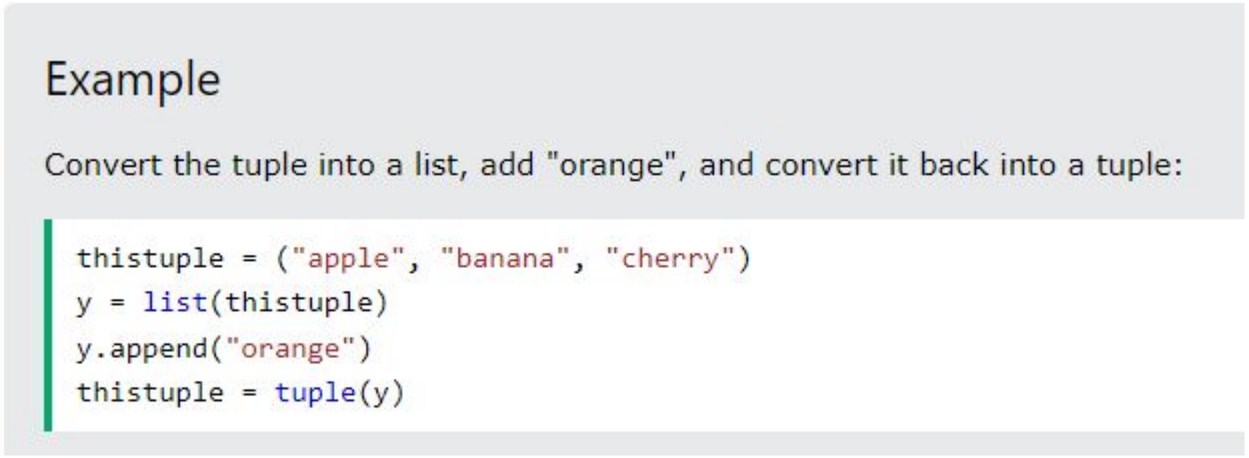
- It is a collection of elements which is ordered and unchangeable.
- **Create a Tuple :**

  Example :

  ```
  fruits = (“apple”, “banana”, “mango”)
  ```

- Accessing a tuple (same as List) by **index operator** [ ]
- **Example** : print(fruits[0])

Update tuple



Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:

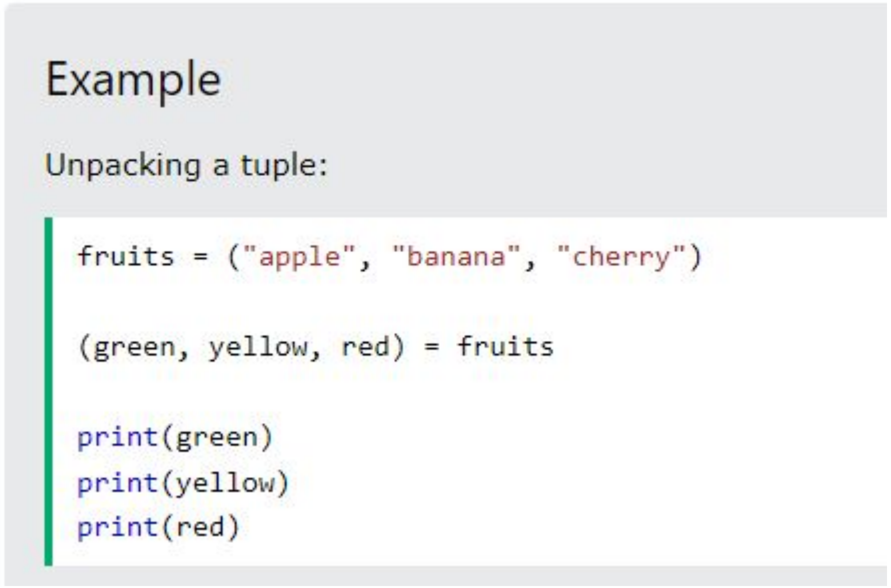
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

Join two Tuples

Example :

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

y = ("orange",)

thistuple += y

print(thistuple)
```

UnPacking Tuple

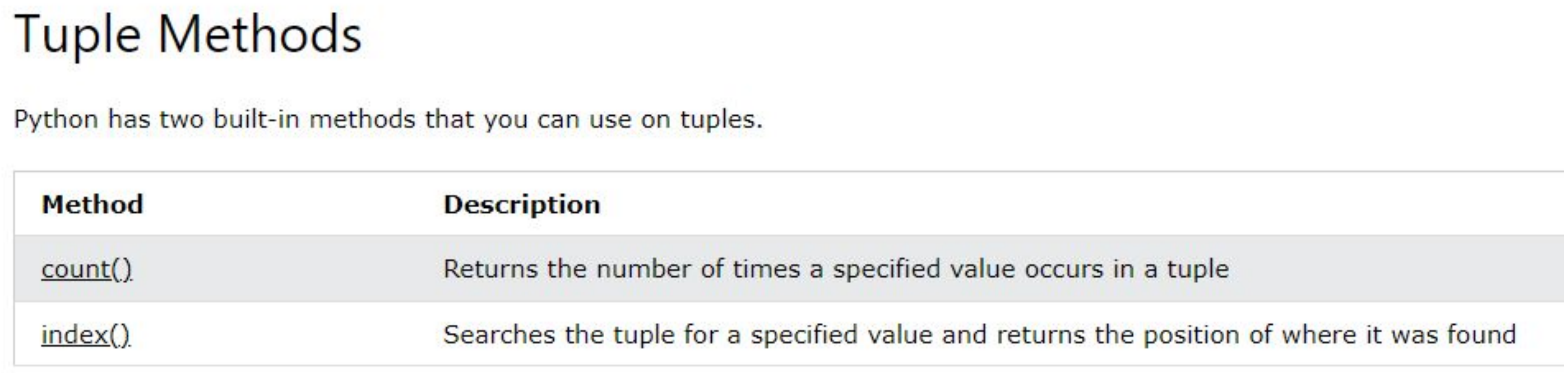


Example

Unpacking a tuple:

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

(green, yellow, red) = fruits

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

Tuple built in Functions



Tuple Methods

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

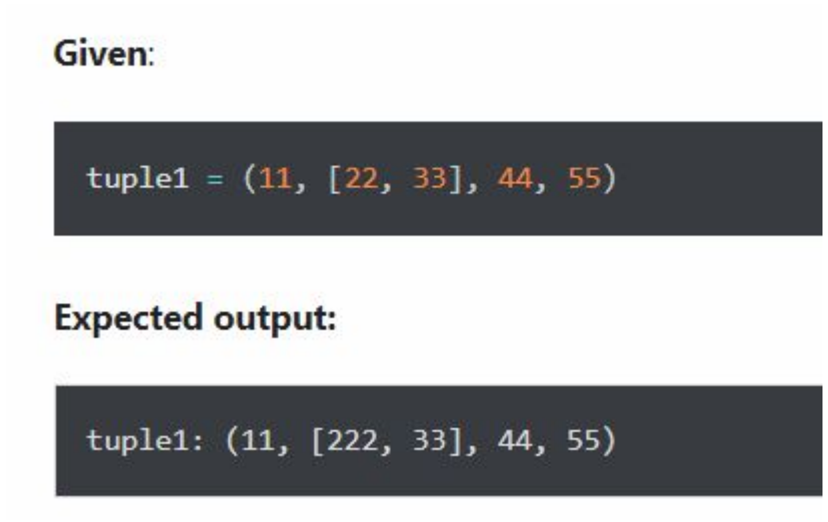
Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Examples

Swapping

```
tuple1 = (11, 22)

tuple2 = (99, 88)

tuple1, tuple2 = tuple2, tuple1

print(tuple2)

print(tuple1)
```

Modify the tuple



Given:

```
tuple1 = (11, [22, 33], 44, 55)
```

Expected output:

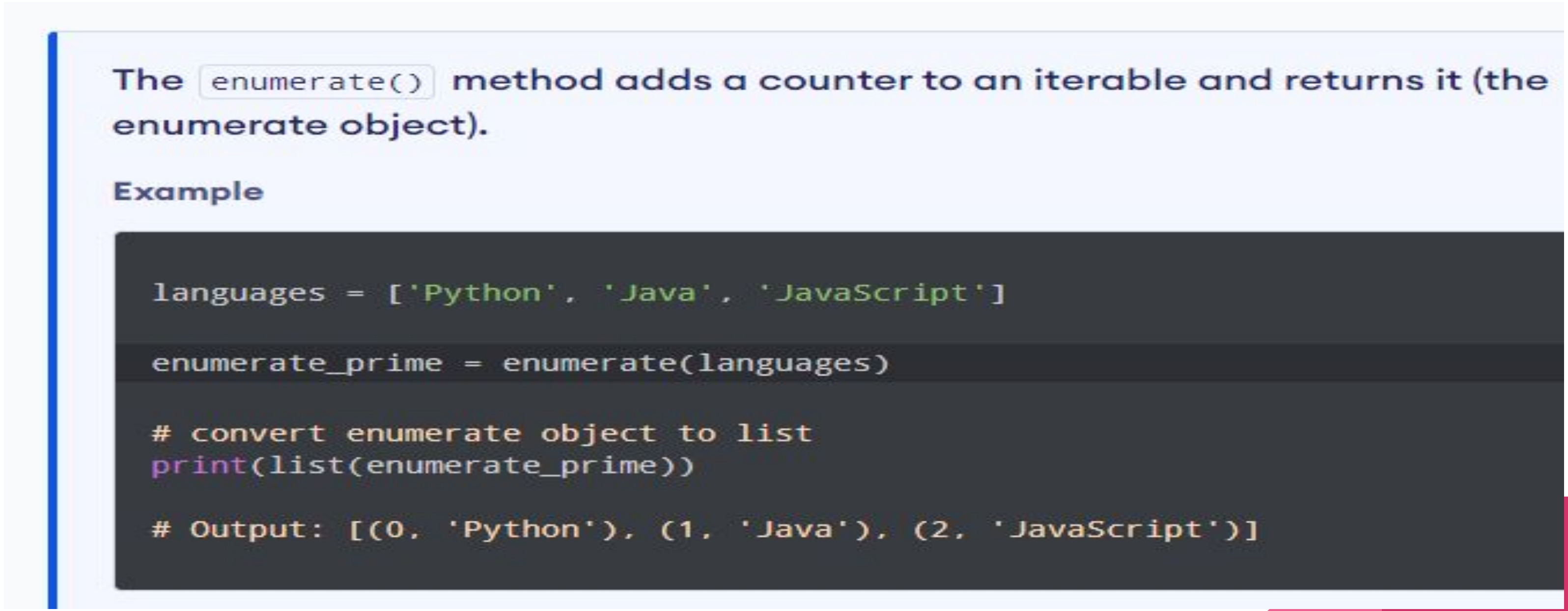
```
tuple1: (11, [222, 33], 44, 55)
```

Solution

```
tuple1 = (11, [22, 33], 44, 55)

tuple1[1][0] = 222

print(tuple1)
```

Enumerate

enumerate() **allows us to iterate through a sequence but it keeps track of both the index and the element**.

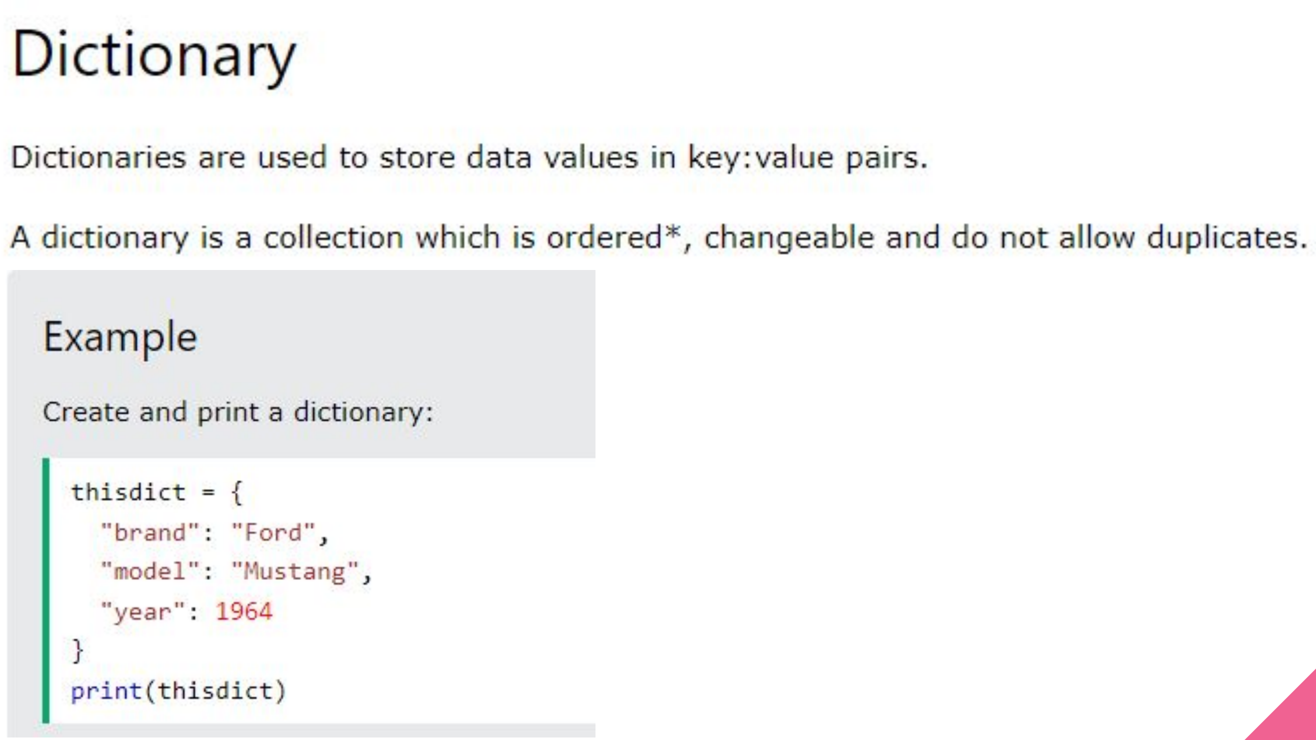


The `enumerate()` method adds a counter to an iterable and returns it (the enumerate object).

Example

```
languages = ['Python', 'Java', 'JavaScript']

enumerate_prime = enumerate(languages)

# convert enumerate object to list
print(list(enumerate_prime))

# Output: [(0, 'Python'), (1, 'Java'), (2, 'JavaScript')]
```

Dictionary in Python


Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

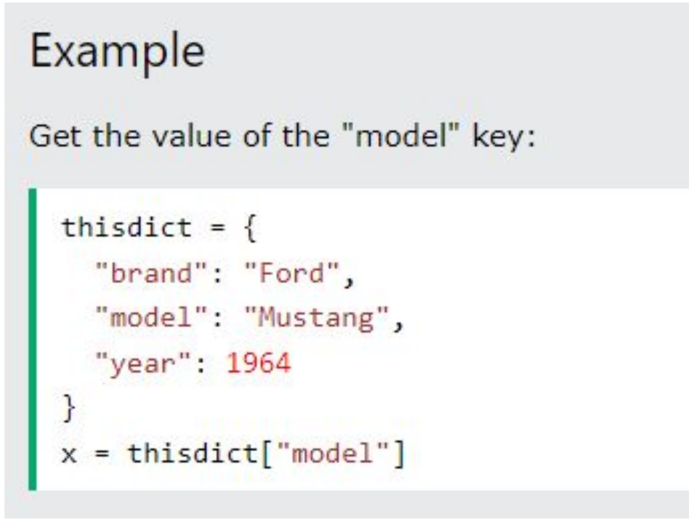
Example

Create and print a dictionary:

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

Accessing Items

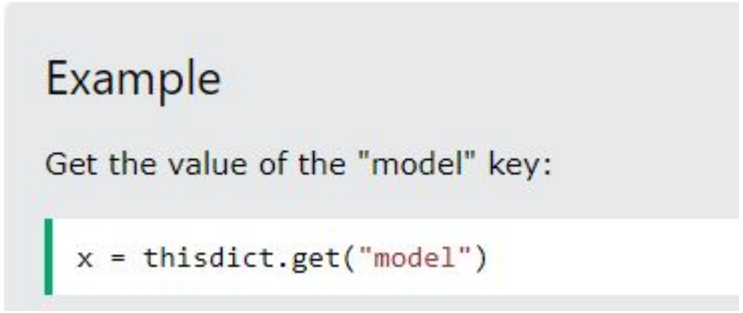
You can access the items of a dictionary by referring to its key name, inside square brackets:



Example

Get the value of the "model" key:

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

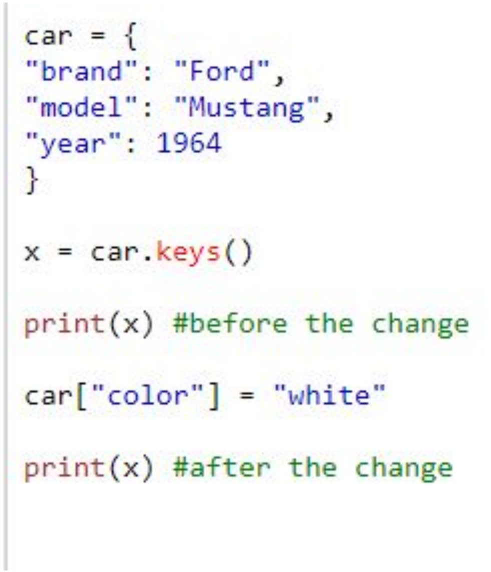


Example

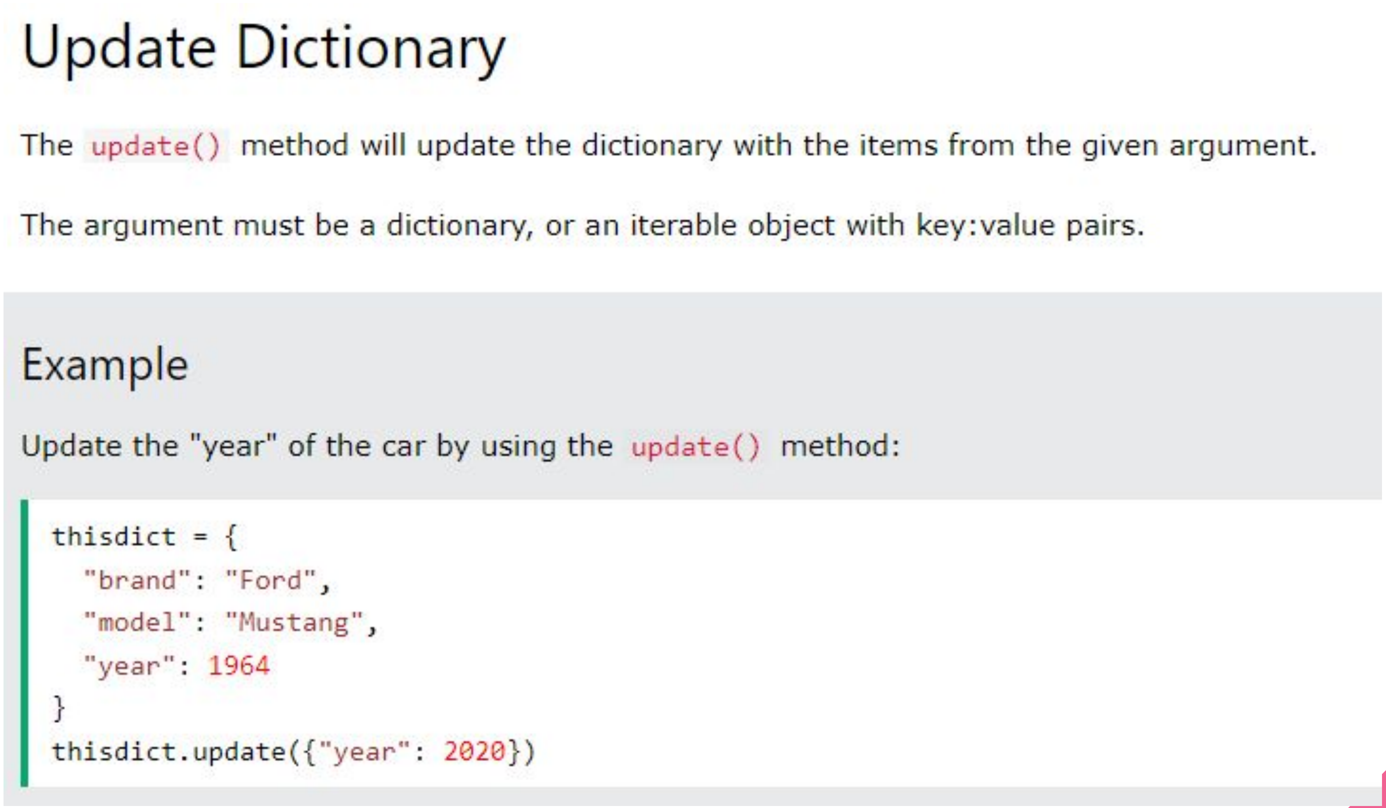
Get the value of the "model" key:

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

Add a value to Dictionary



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

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```



Update Dictionary

The `update()` method will update the dictionary with the items from the given argument.

The argument must be a dictionary, or an iterable object with key:value pairs.

Example

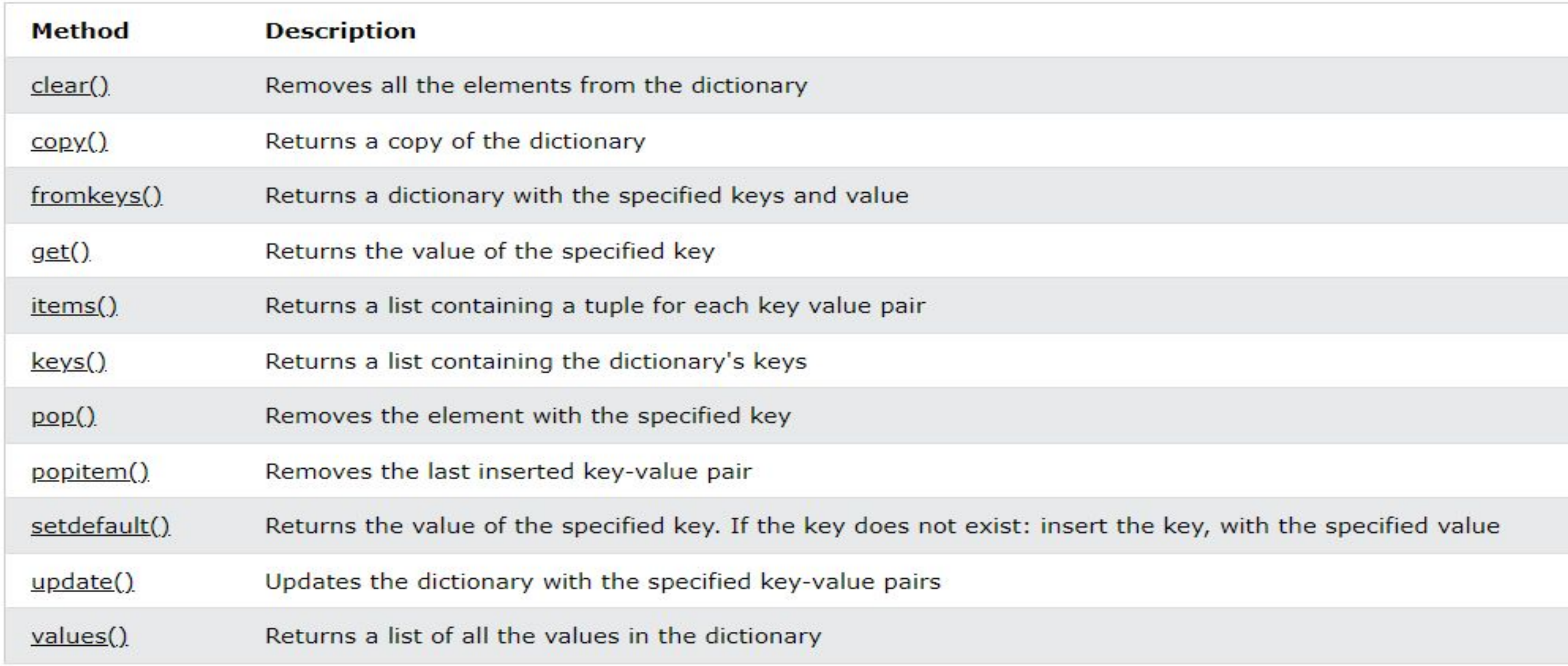
Update the "year" of the car by using the `update()` method:

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

Remove item in Dictionary

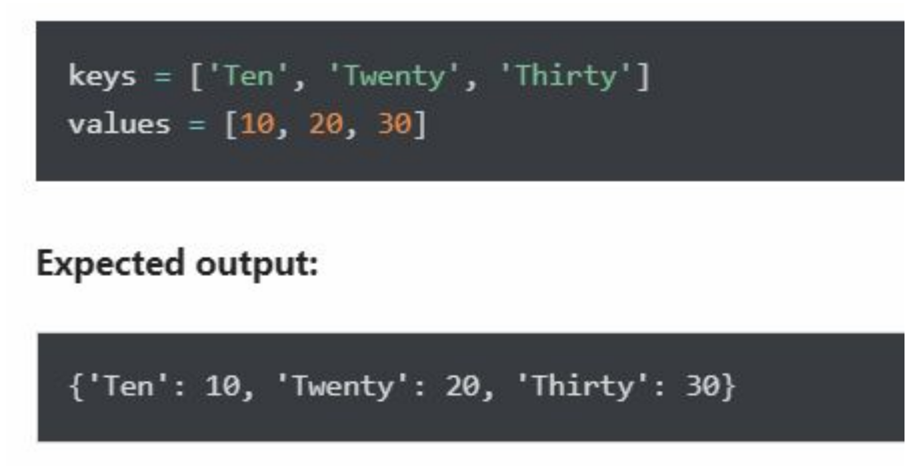
- pop()
- popitem()
- del dict_variable[key]
- clear()

Dictionary Functions



Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Convert two lists into a dictionary



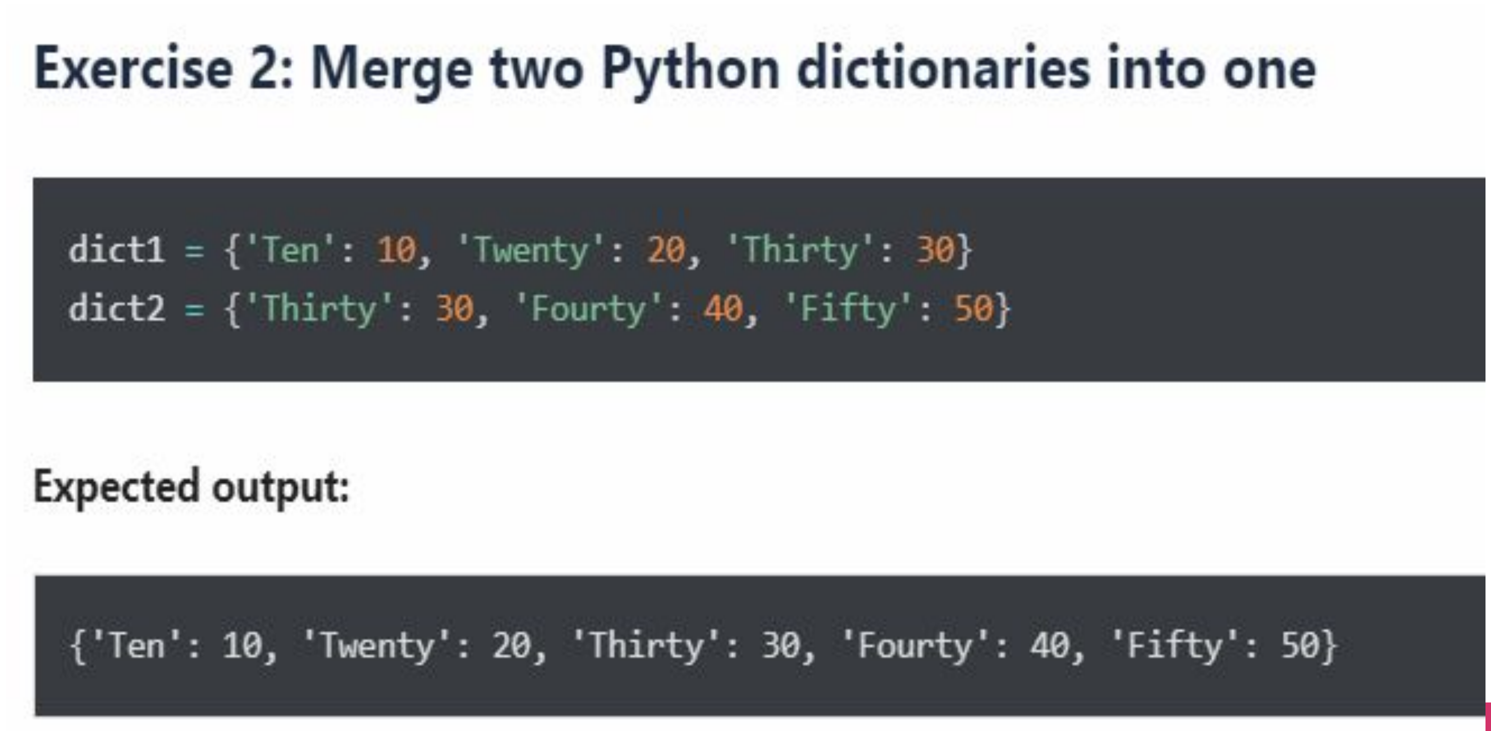
```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30]
```

Expected output:

```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30}
```

Solution

```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30]

# empty dictionary
res_dict = dict()

for i in range(len(keys)):
    res_dict.update({keys[i]: values[i]})
print(res_dict)
```



Exercise 2: Merge two Python dictionaries into one

```python
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30}
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

Expected output:

```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

Solution

```
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30}

dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}



dict3 = dict1.copy()

dict3.update(dict2)

print(dict3)
```