Python Programming

Tuples and Dictionaries

By
Mr. Sumedrao M. Gaikwad
BCA Department
Vivekanand College(Autonomous)
Kolhapur.

Tuples

Tuples are very similar to lists, but they are *immutable* (i.e., unchangeable)

Tuples are written with round brackets as follows:

```
t1 = (1, 2, 3)
t2 = (“a”, “b”, “c”, ”d”)
t3 = (200, “A”, [4, 5], 3.2)
print(t1)
print(t2)
print(t3)
```

Tuples

Like lists, tuples can:

- Contain any and different types of elements
- Contain duplicate elements (e.g., (1, 1, 2))
- Be indexed exactly in the same way (i.e., using the [] brackets)
- Be sliced exactly in the same way (i.e., using the [::] notation)
- Be concatenated (e.g., t = (1, 2, 3) + (“a”, “b”, “c”))
- Be repeated (e.g., t = (“a”, “b”) * 10)
- Be nested (e.g., t = ((1, 2), (3, 4), ((“a”, “b”, ”c”), 3.4))
- Be passed to a function, but will result in *pass-by-value* and not *pass-by-reference* outcome since it is immutable
- Be iterated over

Examples

```
t1 = ("a", "b", "c")
print(t1[::-1])
t2 = ("a", "b", "c")
t3 = t1 + t2
print(t3)
t3 = t3 * 4
print(t3)

for i in t3:
    print(i, end = " ")

print()
```

This will print the elements of t1 in a *reversed order*, but will not change t1 itself since it is immutable

This will concatenate t1 and t2 and assign the result to t3 (again, t1 and t2 will be unchanged since they are immutable)

This will repeat t3 four times and assign the result to t3. Hence, t3 will be *overwritten* (i.e., NOT changed in place- because it is immutable-, but redefined with a new value)

Examples

```
t4 = ((1, 2, 3), ("a", "b", "c"))
for j in t4:
    for k in j:
        print(k,end = " ")
    print()
```

This is an example of nesting, where a matrix with 2 rows and 3 columns is created. The first row includes the elements 1, 2, and 3. The second row includes the elements “a”, “b”, and “c”.

This *outer loop* iterates over each element in t4; that is, it gets first the element (1, 2, 3) and second the element (“a”, “b”, “c”)

This *inner loop* iterates over each element read by the outer loop; that is, it first iterates over the elements of the element (1, 2, 3), and second it iterates over the elements of the element (“a”, “b”, “c”)

Examples

```
def func1(t):
    t = t * 2

t = (1, 2, 3)
print(t)
func1(t)
print(t)
```

This change on *t* remains *local* to the function since a value of *t* was passed and not a *reference* to it

This will output (1, 2, 3)

This will also output (1, 2, 3) since tuples are immutable, hence, will always exhibit a passed-by-value behavior

Using Functions with Tuples

You can also use functions with tuples

```
t1 = (1, 2, 3, 1, 5, 1)
print(t1.count(1))
print(t1.index(1))
```

Output:

3

0

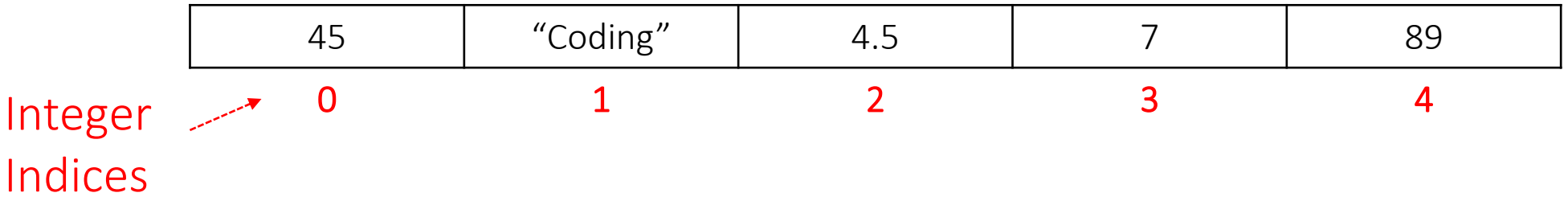
The count(*x*) function returns the number of elements with the specified value *x* (e.g., *x* is 1 in this example)

The index(*x*) function returns the index of the *first element* with the specified value *x* (e.g., x is 1 in this example)

In fact, Python has only these two built-in functions that can be used on tuples

Towards Dictionaries

Lists and tuples hold elements with *only integer* indices



So in essence, each element has an *index* (or a *key*) which can *only* be an integer, and a *value* which can be of any type (e.g., in the above list/tuple, the first element has key 0 and value 45)

What if we want to store elements with non-integer indices (or keys)?

Dictionaries

In Python, you can use a dictionary to store elements with *keys of any types* (not necessarily only integers like lists and tuples) and *values of any types* as well

keys of different types

Values of different types

The above dictionary can be defined in Python as follows:

```
dic = {"NUM":45, 1000:"coding", 2000:4.5, 3.4:7, "XXX":89}
```

key

value

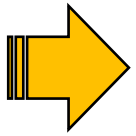
Each element is a *key:value* pair, and elements are separated by commas

Dictionaries

In summary, dictionaries:

- Can contain any and different types of elements (i.e., keys and values)
- Can contain only *unique* keys but duplicate values

```
dic2 = {"a":1, "a":2, "b":2}
print(dic2)
```



Output: {'a': 2, 'b': 2}

The element “a”:2 will override the element “a”:1 because only ONE element can have key “a”

- Can be indexed *but only* through keys (i.e., dic2[“a”] will return 1 but dic2[0] will return an error since there is no element with key 0 in dic2 above)

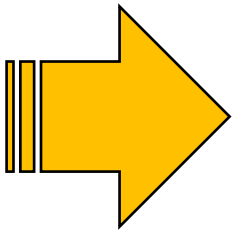
Dictionaries

In summary, dictionaries:

- CANNOT be concatenated
- CANNOT be repeated
- Can be nested (e.g., d = {"first":{1:1}, "second":{2:"a"}}
- Can be passed to a function and will result in a *pass-by-reference* and not *pass-by-value* behavior since it is *immutable* (like lists)

```
def func1(d):
    d["first"] = [1, 2, 3]

dic = {"first":{1:1},
"second":{2:"a"}}
print(dic)
func1(dic)
print(dic)
```



Output:

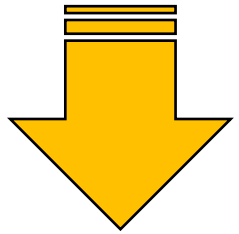
{'first': {1: 1}, 'second': {2: 'a'}}
{'first': [1, 2, 3], 'second': {2: 'a'}}

Dictionaries

In summary, dictionaries:

Can be iterated over

```
dic = {"first": 1, "second": 2, "third": 3}
for i in dic:
    print(i)
```



Output:

first
second
third

ONLY the keys will be returned.

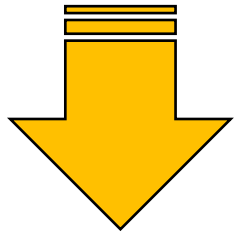
How to get the values?

Dictionaries

In summary, dictionaries:

Can be iterated over

```
dic = {"first": 1, "second": 2, "third": 3}
for i in dic:
    print(dic[i])
```



Output:

1
2
3

Values can be accessed via indexing!

Adding Elements to a Dictionary

How to add elements to a dictionary?

By indexing the dictionary via a key and assigning a corresponding value

```
dic = {"first": 1, "second": 2, "third": 3}
print(dic)
dic["fourth"] = 4
print(dic)
```

Output:

```
{'first': 1, 'second': 2, 'third': 3}
{'first': 1, 'second': 2, 'third': 3, 'fourth': 4}
```

Adding Elements to a Dictionary

How to add elements to a dictionary?

By indexing the dictionary via a key and assigning a corresponding value

```
dic = {"first": 1, "second": 2, "third": 3}
print(dic)
dic[”second"] = 4
print(dic)
```

If the key already exists, the value will be overridden

Output:

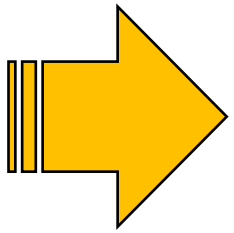
```
{'first': 1, 'second': 2, 'third': 3}
{'first': 1, 'second’: 4, 'third': 3}
```

Deleting Elements to a Dictionary

How to delete elements in a dictionary?

By using **del**

```
dic = {"first": 1, "second": 2, "third": 3}
print(dic)
dic["fourth"] = 4
print(dic)
del dic["first"]
print(dic)
```



Output:

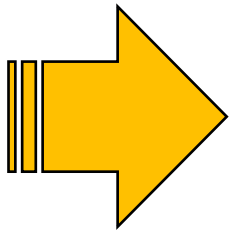
```
{'first': 1, 'second': 2, 'third': 3}
{'first': 1, 'second': 2, 'third': 3, 'fourth': 4}
{'second': 2, 'third': 3, 'fourth': 4}
```

Deleting Elements to a Dictionary

How to delete elements in a dictionary?

Or by using the function pop(key)

```
dic = {"first": 1, "second": 2, "third": 3}
print(dic)
dic["fourth"] = 4
print(dic)
dic.pop("first")
print(dic)
```



Output:

```
{'first': 1, 'second': 2, 'third': 3}
{'first': 1, 'second': 2, 'third': 3, 'fourth': 4}
{'second': 2, 'third': 3, 'fourth': 4}
```

Dictionary Functions

Many other functions can also be used with dictionaries

Function	Description
dic.clear()	Removes all the elements from dictionary dic
dic.copy()	Returns a copy of dictionary dic
dic.items()	Returns a list containing a tuple for each key-value pair in dictionary dic
dic.get(k)	Returns the value of the specified key k from dictionary dic
dic.keys()	Returns a list containing all the keys of dictionary dic
dic.pop(k)	Removes the element with the specified key k from dictionary dic

Dictionary Functions

Many other functions can also be used with dictionaries

Function	Description
dic.popitem()	Removes the last inserted key-value pair in dictionary dic
dic.values()	Returns a list of all the values in dictionary dic