Cheat Sheet

Working with Dictionaries

Dictionary Methods

Python provides dictionary methods that allow us to work with dictionaries.

- copy()
- get()
- update()
- fromkeys() and more..

Let’s learn few among them

Referring Same Dictionary Object

Code

PYTHON

```python
dict_a = {
    'name': 'Teja',
    'age': 15
}
dict_b = dict_a
dict_b['age'] = 20
print(dict_a)
print(id(dict_a))
print(id(dict_b))
```

Output

```
{'name':'Teja', 'age': 20}
140170705626624
140170705626624
```

Copy of Dictionary

`dict.copy()` returns copy of a dictionary.

Code

PYTHON

```python
dict_a = {
  'name': 'Teja',
  'age': 15
}
dict_b = dict_a.copy()
dict_b['age'] = 20
print(dict_a)
print(id(dict_a))
print(id(dict_b))
```

Output

```
{'name':'Teja',  'age': 15}
140664418952704
140664418952896
```

Copy of List

Code

PYTHON

```python
list_a = ['Teja', 15]
list_b = list_a.copy()
list_b.extend([20])
print(list_a)
print(id(list_a))
print(id(list_b))
```

Output

```
['Teja', 15]
139631861316032
139631860589504
```

Operations on Dictionaries

- len()
- clear()
- Membership Check

Code

PYTHON

```python
dict_a = {
'name': 'Teja',
'age': 15
}
print(len(dict_a))  # length of dict_a
if 'name' in dict_a:  # Membership Check
    print("True")
dict_a.clear()  # clearing dict_a
print(dict_a)
```

Output

```
2
True
{}
```

Iterating

Cannot add/remove dictionary keys while iterating the dictionary.

Code

PYTHON

```python
dict_a = {'name': 'Teja', 'age': 15}
for k in dict_a.keys():
    if k == 'name':
        del dict_a[k]
print(dict_a)
```

Output

```
RuntimeError: dictionary changed size during iteration
```

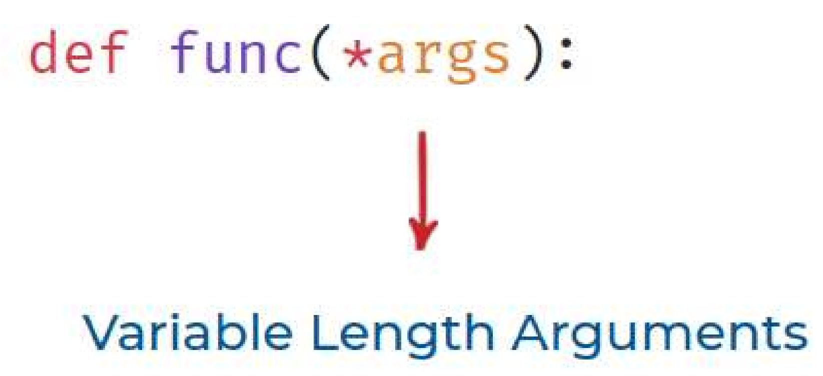
Arbitrary Function Arguments

Passing Multiple Values

We can define a function to receive any number of arguments.

We have already seen such functions

- max(*args) max(1,2,3..)
- min(*args) min(1,2,3..)



```python
def func(*args):
```

Variable Length Arguments

Variable length arguments are packed as tuple.

Code

PYTHON

```python
def more_args(*args):
  print(args)

more_args(1, 2, 3, 4)
more_args()
```

Output

```
(1, 2, 3, 4)
()
```

Unpacking as Arguments

If we already have the data required to pass to a function as a sequence, we can

unpack it with

- while passing.

Code

PYTHON

```python
def greet(arg1="Hi", arg2="Ram"):
    print(arg1 + " " + arg2)

data = ["Hello", "Teja"]
greet(*data)
```

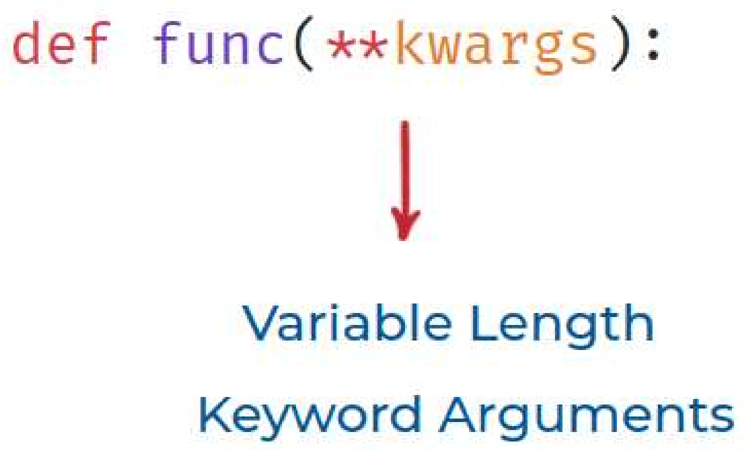
Output

```
Hello Teja
```

Multiple Keyword Arguments

We can define a function to receive any number of keyword arguments.

Variable length kwargs are packed as dictionary.



Code

PYTHON

```python
def more_args(**kwargs):
    print(kwargs)

more_args(a=1, b=2)
more_args()
```

Output

```
{'a': 1, 'b': 2}
{}
```

Iterating

kwargs is a dictionary. We can iterate over them like any other dictionary.

Code

PYTHON

```python
def more_args(**kwargs):
    for i, j in kwargs.items():
      print('{}:{}'.format(i,j))

more_args(a=1, b=2)
```

Output

```
a:1
b:2
```

Unpacking as Arguments

Code - 1

PYTHON

```python
def greet(arg1="Hi", arg2="Ram"):
    print(arg1 + " " + arg2)

data = {'arg1':'Hello', 'arg2':'Teja'}
greet(**data)
```

Output

```
Hello Teja
```

Code - 2

PYTHON

```python
def greet(arg1="Hi", arg2="Ram"):
  print(arg1 + " " + arg2)

data = {'msg':'Hello', 'name':'Teja'}
greet(**data)
```

Output

```
TypeError: greet() got an unexpected keyword argument 'msg'
```

Submit Feedback