# Dictionaries

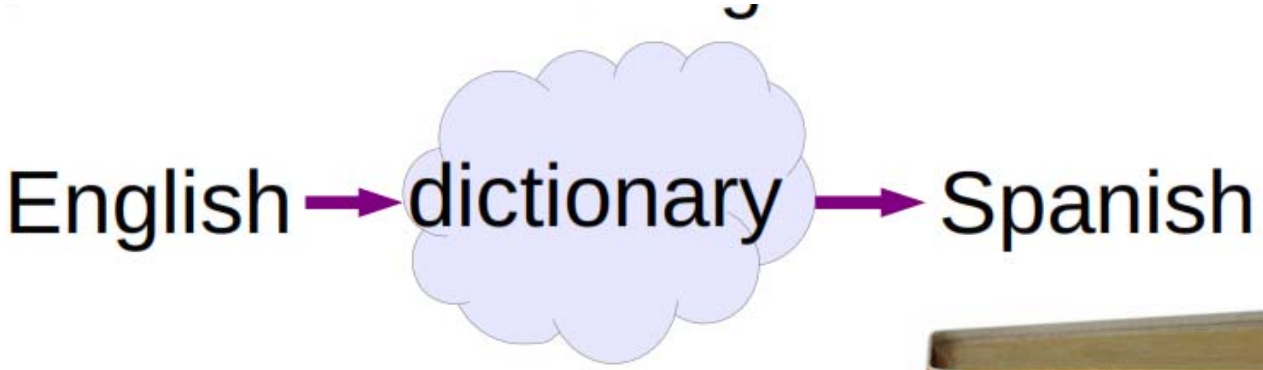
# Dictionaries

- Dictionaries are a type of data structure that are a little like lists in that they are a sequence of elements.
- Each element in a dictionary consists of a key/value pair separated by a colon in a “{“  and “ }”.
- For example “george”:”blue”.
- The string “george” is the key and “blue” is the value.
- Keys can be other datatypes, for example numbers.
- However, the datatype of all keys in a dictionary must be the same.
- Unlike the lists, elements are not accessed using indexes as are lists, but using keys.

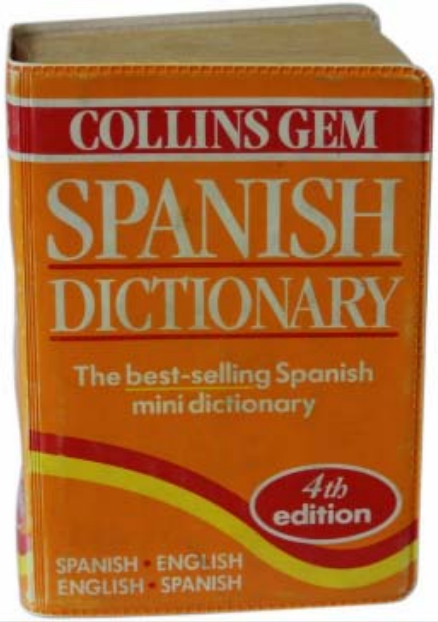
# Dictionaries

- **Dictionaries store a *mapping between a set of keys* and a set of values.**
  - Dictionaries are mutable
  - *Keys can be any immutable type.*
  - Values can be any type
- **Values and keys can be of different types in a single dictionary, You can**
  - define
  - modify
  - view
  - lookup
  - delete
- **the key-value pairs in the dictionary.**

# Dictionaries







# Creation

- **First we will look at creating a dictionary. In the same way that we can create a list with square brackets, we can create a dictionary with curly ones.**

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py

File Edit Format Run Options Window Help

```
en_to_es = { 'cat':'gato' , 'dog':'perro' }
print(en_to_es)


#output
    {'cat': 'gato', 'dog': 'perro'}
```

# Access

- We can index this dictionary by key to fetch and change the keys' associated values.
- The dictionary index operation uses the same syntax as that used for sequences, but the item in the square brackets is a key, not a relative position:

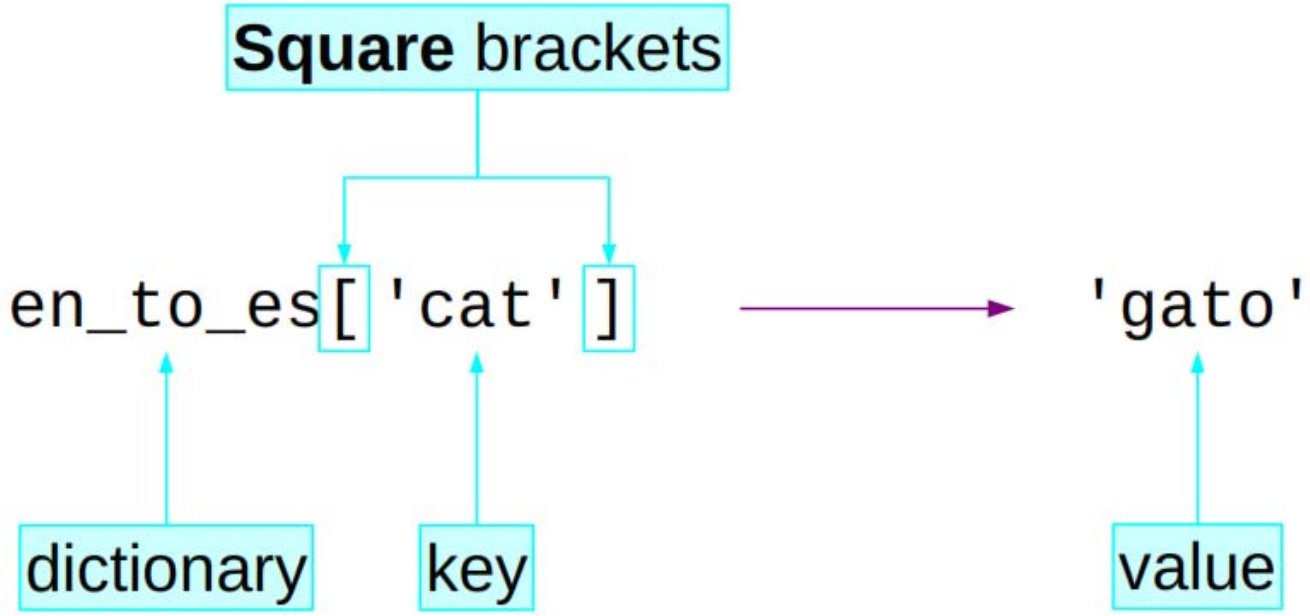
*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4

File Edit Format Run Options Window Help

```
en_to_es = { 'cat':'gato' , 'dog':'perro' }
print(en_to_es['cat'])
print(en_to_es['dog'])


#output
    gato
    perro
```

# Dictionaries



# Access

- We can index this dictionary by key to fetch and change the keys' associated values, but can not use the value.

simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic

File Edit Format Run Options Window Help

```
en_to_es = { 'cat':'gato' , 'dog':'perro' }
print(en_to_es['cat'])
print(en_to_es['dog'])
print(en_to_es['perro'])
```

Output

```
gato
perro
Traceback (most recent call last):
  File "C:/Documents and Settings/admin/D
edic.py", line 4, in <module>
    print(en_to_es['perro'])
KeyError: 'perro'
```

# Adding to a dictionary

- Adding key-value pairs to a dictionary is a lot easier than it is with lists. With lists we needed to append on the end of a list. With dictionaries, because there is no inherent order, we can simply define them with a simple expression on the left hand side.

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
en_to_es = { 'cat':'gato' , 'dog':'perro' }
en_to_es['mouse']='raton'

print(en_to_es)
```

Output

```
{'dog': 'perro', 'cat': 'gato', 'mouse': 'raton'}
```

# Removing from a dictionary

- We can use del to remove from a dictionary just as we did with lists.

simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)

File Edit Format Run Options Window Help

```
en_to_es = { 'cat':'gato' , 'dog':'perro' }
en_to_es['mouse']='raton'
print(en_to_es)
del en_to_es['dog']
print(en_to_es)
```

Output

```
{'cat': 'gato', 'mouse': 'raton', 'dog': 'perro'}
{'cat': 'gato', 'mouse': 'raton'}
```

# Creation from empty

- The following code, for example, starts with an empty dictionary and fills it out one key at a time. Unlike out-of-bounds assignments in lists, which are forbidden, assignments to new dictionary keys create those keys.

simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)

File Edit Format Run Options Window Help

```
D = {}
D['name'] = 'Bob'        # Create keys by assignment
D['job']  = 'dev'
D['age']  = 40
print(D)
D['age']  += 1
print(D)
```

Output

```
{'job': 'dev', 'name': 'Bob', 'age': 40}
{'job': 'dev', 'name': 'Bob', 'age': 41}
```

# Nesting

- Suppose, though, that the information is more complex. Perhaps we need to record a first name and a last name, along with multiple job titles. This leads to another application of Python's object nesting in action.

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
rec = {'name': {'first': 'Bob', 'last': 'Smith'},
       'jobs': ['dev', 'mgr'],
       'age':  40.5}
print(rec)



#output

{'name': {'first': 'Bob', 'last': 'Smith'},
 'age': 40.5, 'jobs': ['dev', 'mgr']}
```

# Nesting

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
rec = {'name': {'first': 'Bob', 'last': 'Smith'},
        'jobs': ['dev', 'mgr'],
        'age':  40.5}
print(rec)
print(rec['name'])
print(rec['name']['last'])
print(rec['jobs'])
print(rec['jobs'][-1])
rec['jobs'].append('janitor')
print(rec)

#output

{'jobs': ['dev', 'mgr'], 'name': {'first': 'Bob', 'last': 'Smith'}, 'age': 40.5}
{'first': 'Bob', 'last': 'Smith'}
Smith
['dev', 'mgr']
mgr
{'jobs': ['dev', 'mgr', 'janitor'], 'name': {'first': 'Bob', 'last': 'Smith'},
 'age': 40.5}
```

# Operators for dictionary

- **k in d :** True, if a key k exists in the dictionary d
- **k not in d:** True, if a key k doesn't exist in the dictionary d

# Dictionary comparisons

```
D1 = {'a':1, 'b':2}
D2 = {'a':1, 'b':3}
if (D1 == D2):
   print('They are equal')
else:
   print('They are not equal')

# D1 < D2   TypeError: unorderable types: dict() < dict()
```

This can be done with dic.items() (see later)

# Built-in Dictionary Functions

- Python includes the following dictionary functions
- len(d) : return the number of stored entries, i.e. the number of (key, value) pairs
- type(d): return the type dictionary
- del d[k] : delets the key k together with his vlaue

# All and Any

- All: Return True if all elements of the dict are true (or if the dict is empty).
- Any: Return True if any element of the dict true. If the dict is empty, return False.

simpledic.py - /home/nowzari/Desktop/python/python-my/python/examples/18-dic/simpl

File Edit Format Run Options Window Help

```
plants = {}
if all(plants):
    print('all element of an empty plants are true')
else:
    print('all element of an empty plants are not true')

plants = {"radish": 2, "squash": 4, "carrot": 7, "":5}
if all(plants):
    print('all element of plants are true')
else:
    print('all element of plants are not true')

plants = {}
if any(plants):
    print('empty plants has a true element')
else:
    print('empty plants do not has any true element')

plants = {"radish": 2, "squash": 4, "carrot": 7, "":5}
if any(plants):
    print('plants has a true element')
else:
    print('plants do not has a true element')
```

```
all element of an empty plants are true
all element of plants are not true
empty plants do not has any true element
plants has a true element
```

# sorted

sorted.py - /mnt/3494FC2794FBE8EE/nowzari/awork/course/course/python/python-my/python/examp

File Edit Format Run Options Window Help

```
bobRec = {"name":'Bob', "job":'dev', "age":40}
slk=sorted(bobRec)
print(slk)
```

```
['age', 'job', 'name']
```

# Enumerate

- This is a built-in method. Enumerate() returns a tuple of an index and the element value at that index. It is often used on a list.

*dicenum.py - /home/nowzari/Desktop/python/python-my/python/exam

File Edit Format Run Options Window Help

```
M={"class0":1, "class1":2, "class2":3}
G=list(enumerate(M))
print(G)

#oputput

  [(0, 'class0'), (1, 'class2'), (2, 'class1')]
```

# Type Conversion functions

# str

- The method str() produces a printable string representation of a dictionary.

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
rec = {'name': {'first': 'Bob', 'last': 'Smith'},
        'jobs': ['dev', 'mgr'],
        'age':  40.5}
st=str(rec)
print(st)

#output
{'age': 40.5, 'jobs': ['dev', 'mgr'], 'name': {'last': 'Smith', 'first': 'Bob'}}
```

# dict

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
bobRec = dict(name='Bob', job='dev', age=40)
print(bobRec)

#output

{'job': 'dev', 'age': 40, 'name': 'Bob'}
```

# list

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.p

File Edit Format Run Options Window Help

```
bobRec = dict(name='Bob', job='dev', age=40)
ks=list(bobRec)
print(ks)


#output

['age', 'name', 'job']
```

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py

File Edit Format Run Options Window Help

```
bobRec = dict(name='Bob', job='dev', age=40)
slk=sorted(bobRec)
for key in slk:
    print(key, '=>', bobRec[key])

#output

age => 40
job => dev
name => Bob
```

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic

File Edit Format Run Options Window Help

```
bobRec = dict(name='Bob', job='dev', age=40)

for key in sorted(bobRec):
    print(key, '=>', bobRec[key])

#output

age => 40
job => dev
name => Bob
```

# Dictionary creation by Enumerate

```
L = ['a', 'b', 'c', 'd']
D={letter: i for i,letter in enumerate(L, start=1)}
print(D)


#output:

{'b': 2, 'c': 3, 'd': 4, 'a': 1}
```

# Dictionary construction with zip

```
keys = ['spam', 'eggs', 'toast']
vals = [1, 3, 5]
print(list(zip(keys, vals)))

D2 = {}
for (k, v) in zip(keys, vals): D2[k] = v
print(D2)

keys = ['spam', 'eggs', 'toast']
vals = [1, 3, 5]
D3 = dict(zip(keys, vals))
print(D3)

#output:
[('spam', 1), ('eggs', 3), ('toast', 5)]
{'toast': 5, 'spam': 1, 'eggs': 3}
{'toast': 5, 'spam': 1, 'eggs': 3}
```

# Dictionary construction with zip

```
l = ['a', 'b', 'c', 'd']
Li={ x:(y+1) for (x,y) in zip(l, range(len(l))) }
print(Li)

#output:

{'a': 1, 'd': 4, 'c': 3, 'b': 2}
```

# Type specific functions for dictionaries

- There are some type specific functions for **dictionaries**, such as: clear, copy, and type conversion methods.

# clear

- The method clear() removes all items from the dictionary.

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3

File Edit Format Run Options Window Help

```
mydict = {'Name': 'Zara', 'Age': 7};

print ("Start Len : %d" %  len(mydict))
mydict.clear()
print ("End Len : %d" %  len(mydict))

#output

Start Len : 2
End Len : 0
```

# copy

- The method copy() returns a shallow copy of the dictionary.

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = dict1.copy()
dict2['Name']='ali'
print ("Dictinary one: ", dict1)
print ("Dictinary two: ", dict2)

print('.....................................')

dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = dict1
dict2['Name']='ali'
print ("Dictinary one: ", dict1)
print ("Dictinary two: ", dict2)


#output

Dictinary one:  {'Name': 'Zara', 'Age': 7}
Dictinary two:  {'Name': 'ali', 'Age': 7}
.....................................
Dictinary one:  {'Name': 'ali', 'Age': 7}
Dictinary two:  {'Name': 'ali', 'Age': 7}
```

# keys

- The method keys() returns a list of all the available keys in the dictionary.

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simp

File Edit Format Run Options Window Help

```
mydict = {'Name': 'Zara', 'Age': 7}
lst=  mydict.keys()
print ("Keys : ", lst)
lst=  list(mydict.keys())
print ("Keys : ", lst)

#output

Keys :  dict_keys(['Name', 'Age'])
Keys :  ['Name', 'Age']
```

# keys

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
eng2sp = {"one": "uno", "two": "dos", "three": "tres"}
for k in eng2sp.keys():    # The order of the k's is not defined
   print("Got key", k, "which maps to value", eng2sp[k])
ks = list(eng2sp.keys())
print(ks)
for k in eng2sp:
   print("Got key", k)

#output

Got key one which maps to value uno
Got key three which maps to value tres
Got key two which maps to value dos
['one', 'three', 'two']
Got key one
Got key three
Got key two
```

# values

- The method values() returns a list of all the values available in a given dictionary.

**Python program that uses keys**

```
hits = {"home": 125, "sitemap": 27, "about": 43}
keys = hits.keys()
values = hits.values()

print("Keys:")
print(keys)
print(len(keys))

print("Values:")
print(values)
print(len(values))
```

**Output**

```
Keys:
dict_keys(['home', 'about', 'sitemap'])
3
Values:
dict_values([125, 43, 27])
3
```

# get

- The method get() returns a value for the given key. If key is not available then returns default value None.

- dict.get(key, default=None)
  - key -- This is the Key to be searched in the dictionary.
  - default -- This is the Value to be returned in case key does not exist.

# get

**Python program that gets values**

```
plants = {}

# Add three key-value tuples to the dictionary.
plants["radish"] = 2
plants["squash"] = 4
plants["carrot"] = 7

# Get syntax 1.
print(plants["radish"])

# Get syntax 2.
print(plants.get("tuna"))
print(plants.get("tuna", "no tuna found"))
```

**Output**

```
2
None
no tuna found
```

# get

**Python program that causes KeyError**

```
lookup = {"cat": 1, "dog": 2}

# The dictionary has no fish key!
print(lookup["fish"])
```

**Output**

```
Traceback (most recent call last):
  File "C:\programs\file.py", line 5, in <module>
    print(lookup["fish"])
KeyError: 'fish'
```

# get

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
mydict = {'Name': 'Zara', 'Age': 7}
val=mydict.get('Class', 4)
print(val)

val=mydict['Class'] if 'Class' in mydict else 4

#output

4
```

```
def get(key, default=None):
    if key in d:
        return d[k]
    else:
        return default
```

# setdefault

- The method setdefault() is similar to get(), but will set *dict[key]=default* if key is not already in dict.

- **dict.setdefault(key, default=None)**
  - key -- This is the key to be searched.
  - default -- This is the Value to be returned in case key is not found.

# setdefault

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.

File Edit Format Run Options Window Help

```
dict = {'Name': 'Zara', 'Age': 7}

print ("Value :" ,  dict.setdefault('Age', None))
print ("Value :" ,  dict.setdefault('Class', 4))
print(dict)

#output

Value : 7
Value : 4
{'Age': 7, 'Class': 4, 'Name': 'Zara'}
```

# Fromkeys

- This method receives a sequence of keys, such as a list. It creates a dictionary with each of those keys. We can specify a value as the second argument.
- **dict.fromkeys(seq[, value]))**
  - seq -- This is the list of values which would be used for dictionary keys preparation.
  - value -- This is optional, if provided then value would be set to this value

# Fromkeys

**Python that uses fromkeys**

```
# A list of keys.
keys = ["bird", "plant", "fish"]

# Create dictionary from keys.
d = dict.fromkeys(keys, 5)

# Display.
print(d)
```

**Output**

```
{'plant': 5, 'bird': 5, 'fish': 5}
```

# Fromkeys

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4

File Edit Format Run Options Window Help

```
# A list of keys.
keys = ["bird", "plant", "fish"]

# Create dictionary from keys.
d = dict.fromkeys(keys)

# Display.
print(d)

#output

{'fish': None, 'bird': None, 'plant': None}
```

# Items

- With this method we receive a list of two-element tuples. Each tuple contains, as its first element, the key. Its second element is the value.

**Python that uses items method**

```
rents = {"apartment": 1000, "house": 1300}

# Convert to list of tuples.
rentItems = rents.items()

# Loop and display tuple items.
for rentItem in rentItems:
    print("Place:", rentItem[0])
    print("Cost:", rentItem[1])
    print("")
```

**Output**

```
Place: house
Cost: 1300

Place: apartment
Cost: 1000
```

# Items

- We cannot assign elements in the tuples. If you try to assign rentItem[0] or rentItem[1], you will get an error. This is the error message.
- Python error:
- TypeError: 'tuple' object does not support item assignment

# Items

**Python that unpacks items**

```
# Create a dictionary.
data = {"a": 1, "b": 2, "c": 3}

# Loop over items and unpack each item.
for k, v in data.items():
    # Display key and value.
    print(k, v)
```

**Output**

```
a 1
c 3
b 2
```

# Dictionary comparisons

```
D1 = {'b':2, 'a':1}
D2 = {'b':3, 'a':1}
if (D1 == D2):
   print('They are equal')
else:
   print('They are not equal')

# D1 < D2   TypeError: unorderable types: dict() < dict()

L1=sorted(D1.items())
print(L1)
L2=sorted(D2.items())
print(L2)
if (L1 > L2):
   print('L1 is grather that L2')
else:
   print('L1 is less that L2')
```

```
They are not equal
[('a', 1), ('b', 2)]
[('a', 1), ('b', 3)]
L1 is less that L2
```

# Update

- With this method we change one dictionary to have new values from a second dictionary. Update() also modifies existing values. Here we create two dictionaries.

**Python that uses update**

```
# First dictionary.
pets1 = {"cat": "feline", "dog": "canine"}

# Second dictionary.
pets2 = {"dog": "animal", "parakeet": "bird"}

# Update first dictionary with second.
pets1.update(pets2)

# Display both dictionaries.
print(pets1)
print(pets2)
```

**Output**

```
{'parakeet': 'bird', 'dog': 'animal', 'cat': 'feline'}
{'dog': 'animal', 'parakeet': 'bird'}
```

# pop

- Used to remove an item from a dictionary and return its associated value
- **d.pop(key[, default])**
- d[key] if key is in d. If key is not in d but default is specified, the default value is returned instead.
- KeyError if key is not in dictionary and no default is specified

# pop

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py

File Edit Format Run Options Window Help

```
my_dictionary = {'hello': 1, 'goodbye': 2}
hello_value = my_dictionary.pop('hello')
print(hello_value)

print(my_dictionary)

#output

 1
 {'goodbye': 2}
```

# Pop with default value

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.

File Edit Format Run Options Window Help

```
my_dictionary = {'hello': 1, 'goodbye': 2}
foo_value = my_dictionary.pop('foo', None)
print(foo_value)
print(my_dictionary)

#output
None
{'goodbye': 2, 'hello': 1}
```

# popitem

- Pop (i.e. delete and return) a random element from the dictionary
- **Return a  (key, value) tuple if d is not empty.**
- KeyError if d is empty. I personally think that's a stupid exception to raise since no key was ever specified, but, hey, I didn't write the language.

# popitem

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4

File Edit Format Run Options Window Help

```
plants = {"radish": 2, "squash": 4, "carrot": 7}

k, v= plants.popitem()
print(k, v)

#output

carrot 7
```

# Shared References

num-simple.py - /home/nowzari/Desktop/python/pyth

File Edit Format Run Options Window Help

```
X=42
Y=42
print(X==Y)
print(X is Y)

print("id X is", id(X))
print("id Y is", id(Y))

X=40
Y=X
print(X==Y)
print(X is Y)

print("id X is", id(X))
print("id Y is", id(Y))
X=40
Y=5
print(X==Y)
print(X is Y)

print("id X is", id(X))
print("id Y is", id(Y))
```

```
True
True
id X is 10915680
id Y is 10915680
True
True
id X is 10915616
id Y is 10915616
False
False
id X is 10915616
id Y is 10914496
```

# Shared References

simple.py - /home/nowzari/Desktop/python/python

File Edit Format Run Options Window Help

```
L='banana'
M='banana'
print(L==M)
print(L is M)
print(id(L), id(M))    # Same values

L='banana'
M=L
print(L==M)
print(L is M)
print(id(L), id(M))    # Same values

L='banana'
M=L[:]
print(L==M)
print(L is M)
print(id(L), id(M))     # Same values
```

```
True
True
140495238079072 140495238079072
True
True
140495238079072 140495238079072
True
True
140495238079072 140495238079072
```

# Shared References

```
L = [1, 2, 3]
M = [1, 2, 3]
print(L == M)                # same values
print(L is M)                # same object
print(id(L), id(M))          # M and L reference the different object

L = [1, 2, 3]
M = L
print(L == M)                # same values
print(L is M)                # same object
print(id(L), id(M))          # M and L reference the same object

L = [1, 2, 3]
M = L[:]
print(L == M)                # same values
print(L is M)                # same object
print(id(L), id(M))          # M and L rei
```

```
True
False
139638953740936 139639036808200
True
True
139638953741192 139638953741192
True
False
139639036807496 139638953741256
```

# Share Reference

```
L = (1, 2, 3)
M = (1, 2, 3)
print(L == M)                # same values
print(L is M)                # different object
print(id(L), id(M))          # M and L reference the different object

L = (1, 2, 3)
M = L
print(L == M)                # same values
print(L is M)                # same object
print(id(L), id(M))          # M and L reference the same object

L = (1, 2, 3)
M = L[:]
print(L == M)                # same values
print(L is M)                # same obj
print(id(L), id(M))          # M and L
```

```
True
False
140656957100320 140656991084408
True
True
140657039882208 140657039882208
True
True
140656957026376 140656957026376
```

# Shared References

```
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Zara', 'Age': 7};
print (id(dict1), id(dict2))     # different object
dict2['Name']='ali'
print ("Dictinary one: ", dict1)
print ("Dictinary two: ", dict2)
```

```
140457412732232 140457396810632
Dictinary one:  {'Age': 7, 'Name': 'Zara'}
Dictinary two:  {'Age': 7, 'Name': 'ali'}
```

```
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = dict1
print (id(dict1), id(dict2))    # same object
dict2['Name']='ali'
print ("Dictinary one: ", dict1)
print ("Dictinary two: ", dict2)
```

```
140457388081800 140457388081800
Dictinary one:  {'Age': 7, 'Name': 'ali'}
Dictinary two:  {'Age': 7, 'Name': 'ali'}
```

```
dict1 = {'Name': 'Zara', 'Age': 7}
dict2 = {'Name': 'Zara', 'Age': 7}
print(id(dict1['Name']), id(dict2['Name']))  # same object
```

```
140457388089952 140457388089952
```

# Dic as a function’s parameter

**If you pass dictionary  to a function, the passing acts like call-by-reference. Like lists,  we have to consider two cases: Elements of a dictionary can be changed in place, i.e. the dictionary will be changed even in the caller's scope. If a new dictionary is assigned to the name, the old dictionary will not be affected, i.e. the dictionary in the caller's scope will remain untouched.**

*dic-par.py - C:\Documents and Settings\admin\Desktop\intro-python\examples-2\18-dic\dic-par.py (3.4.4)*

File Edit Format Run Options Window Help

```
def func(book):
    book['Tree of life'] = '1987'
    print('----------------------------')
    print('set in the function as pairs: ')
    for (key, value) in book.items():
        print(key, '-> ', value)

    return

table = {'Holy Grail':          '1975',
         'Life of Brian':       '1979',
          'The Meaning of Life': '1983'}

print('dictionary befor function call: ')
for (key, value) in table.items():
    print(key, '-> ', value)

func(table)

print('----------------------------')
print('dictionary after function call: ')
for (key, value) in table.items():
    print(key, '-> ', value)
```

# Dic as a function's parameter

```
dictionary befor function call:
The Meaning of Life ->  1983
Life of Brian ->  1979
Holy Grail ->  1975
----------------------------
set in the function as pairs:
The Meaning of Life ->  1983
Life of Brian ->  1979
Holy Grail ->  1975
Tree of life ->  1987
----------------------------
dictionary after function call:
The Meaning of Life ->  1983
Life of Brian ->  1979
Holy Grail ->  1975
Tree of life ->  1987
```

# Dic as a function's parameter

```
def func(book):
    book={'Holy Grail':           '2000',
          'Life of Brian':        '2001',
           'The Meaning of Life': '2002'}
    print('----------------------------')
    print('set in the function as pairs: ')
    for (key, value) in book.items():
        print(key, '-> ', value)

    return


table = {'Holy Grail':           '1975',
         'Life of Brian':        '1979',
          'The Meaning of Life': '1983'}

print('dictionary befor function call: ')
for (key, value) in table.items():
    print(key, '-> ', value)
func(table)
print('----------------------------')
print('dictionary after function call: ')
for (key, value) in table.items():
    print(key, '-> ', value)
```

# Dic as a function's parameter

```
dictionary befor function call:
The Meaning of Life ->  1983
Holy Grail ->  1975
Life of Brian ->  1979
-----------------------------
set in the function as pairs:
The Meaning of Life ->  2002
Holy Grail ->  2000
Life of Brian ->  2001
-----------------------------
dictionary after function call:
The Meaning of Life ->  1983
Holy Grail ->  1975
Life of Brian ->  1979
```

# Matrix with dictionary

- We can create a n dimensional matrix of size n × m.

```
d = {
    0: {0: 'a', 1: 'b', 2: 'c'},
    1: {0: 'd', 1: 'e', 2: 'f'},
    2: {0: 'g', 1: 'h', 2: 'i'},
}
for i in range(len(d)):
   print(d[i])

#output:

{0: 'a', 1: 'b', 2: 'c'}
{0: 'd', 1: 'e', 2: 'f'}
{0: 'g', 1: 'h', 2: 'i'}
```

# Dictionary with Tuple as index

```
Matrix = {}
Matrix[(2, 3, 4)] = 88
Matrix[(7, 8, 9)] = 99
Matrix[(1, 2, 3)] = 77

X = 2; Y = 3; Z = 4
print(Matrix[(X, Y, Z)])

print(Matrix)

#output:

88
{(2, 3, 4): 88, (7, 8, 9): 99, (1, 2, 3): 77}
```

# Counts letters

**Python that counts letter frequencies**

```
# The first three letters are repeated.
letters = "abcabcdefghi"

frequencies = {}
for c in letters:
    # If no key exists, get returns the value 0.
    # ... We then add one to increase the frequency.
    # ... So we start at 1 and progress to 2 and then 3.
    frequencies[c] = frequencies.get(c, 0) + 1

for f in frequencies.items():
    # Print the tuple pair.
    print(f)
```

**Output**

```
('a', 2)
('c', 2)
('b', 2)
('e', 1)
('d', 1)
('g', 1)
('f', 1)
('i', 1)
('h', 1)
```

# Counting words

*simpledic.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/20-1/simpledic.py (3.4.4)*

File Edit Format Run Options Window Help

```
words = ['the','cat','sat','on','the','mat', 'on', 'the']
counts = {}
for word in words:
    if word in counts:
        counts[word] += 1
    else:
        counts[word] = 1

print(counts)

items = list(counts.items())
items.sort()
for (key, value) in items:
    print(key, value)

#output

{'cat': 1, 'sat': 1, 'mat': 1, 'the': 3, 'on': 2}
cat 1
mat 1
on 2
sat 1
the 3
```

# Search

```
table = {'Holy Grail':           '1975',
         'Life of Brian':        '1979',
          'The Meaning of Life': '1983'}

print(table['Holy Grail'])
print(list(table.items()))      # Value=>Key (year=>title)

Z=[title for (title, year) in table.items() if year == '1975']
print(Z)

#output:

1975
[('Holy Grail', '1975'), ('The Meaning of Life', '1983'), ('Life of Brian', '1979')]

['Holy Grail']
```

# Remove duplicate

```
data = [
  {'id': 10, 'name': 'ali'},
  {'id': 11, 'name': 'hasan'},
  {'id': 12, 'name': 'kazem'},
  {'id': 10, 'name': 'gholi'},
  {'id': 11, 'name': 'kamran'},
]
unique_data = []
for d in data:
    data_exists = False
    for ud in unique_data:
        if ud['id'] == d['id']:
          data_exists = True
          break
    if not data_exists:
        unique_data.append(d)

for d in unique_data:
   print(d)


#output:

{'id': 10, 'name': 'ali'}
{'id': 11, 'name': 'hasan'}
{'id': 12, 'name': 'kazem'}
```

```python
data = [
  {'id': 10, 'name': 'ali'},
  {'id': 11, 'name': 'hasan'},
  {'id': 12, 'name': 'kazem'},
  {'id': 10, 'name': 'gholi'},
  {'id': 11, 'name': 'kamran'},
]
unique_data = []
for d in data:
    data_exists = False
    for ud in unique_data:
        if ud['id'] == d['id']:
          data_exists = True
          break
    if not data_exists:
        unique_data.append(d)

for d in unique_data:
   print(d)

unique_data={ d['id']:d for d in data }.values()
for d in unique_data:
   print(d)

#output:

{'name': 'ali', 'id': 10}
{'name': 'hasan', 'id': 11}
{'name': 'kazem', 'id': 12}

{'name': 'gholi', 'id': 10}
{'name': 'kamran', 'id': 11}
{'name': 'kazem', 'id': 12}
```

# Phone Book

```
def print_menu():
    print('1. Print Phone Numbers')
    print('2. Add a Phone Number')
    print('3. Remove a Phone Number')
    print('4. Lookup a Phone Number')
    print('5. Quit')
    print()

numbers = {}
menu_choice = 0
print_menu()
while menu_choice != 5:
    menu_choice = int(input("Type in a number (1-5): "))
    if menu_choice == 1:
        print("Telephone Numbers:")
        for x in numbers.keys():
            print("Name: ", x, "\tNumber:", numbers[x])
        print()
    elif menu_choice == 2:
        print("Add Name and Number")
        name =input("Name: ")
        phone = input("Number: ")
        numbers[name] = phone
    elif menu_choice == 3:
        print("Remove Name and Number")
        name =input("Name: ")
        if name in numbers:
            del numbers[name]
        else:
            print(name, "was not found")
    elif menu_choice == 4:
        print("Lookup Number")
        name = input("Name: ")
        if name in numbers:
            print("The number is", numbers[name])
        else:
            print(name, "was not found")
    elif menu_choice != 5:
        print_menu()
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Quit

Type in a number (1-5): 2
Add Name and Number
Name: Joe
Number: 545-4464
Type in a number (1-5): 2
Add Name and Number
Name: Jill
Number: 979-4654
Type in a number (1-5): 2
Add Name and Number
Name: Fred
Number: 132-9874
Type in a number (1-5): 1
Telephone Numbers:
Name: Jill      Number: 979-4654
Name: Joe       Number: 545-4464
Name: Fred      Number: 132-9874

Type in a number (1-5): 4
Lookup Number
Name: Joe
The number is 545-4464
Type in a number (1-5): 3
Remove Name and Number
Name: Fred
Type in a number (1-5): 1
Telephone Numbers:
Name: Jill      Number: 979-4654
Name: Joe       Number: 545-4464

Type in a number (1-5): 5
```

# Student Grades

```python
max_points = [25, 25, 50, 25, 100]
assignments = ['hw ch 1', 'hw ch 2', 'quiz    ', 'hw ch 3', 'test']
students = {'#Max': max_points}

def print_menu():
    print("1. Add student")
    print("2. Remove student")
    print("3. Print grades")
    print("4. Record grade")
    print("5. Print Menu")
    print("6. Exit")

def print_all_grades():
    print('\t', end=' ')
    for i in range(len(assignments)):
        print(assignments[i], '\t', end=' ')
    print()
    keys = list(students.keys())
    keys.sort()
    for x in keys:
        print(x, '\t', end=' ')
        grades = students[x]
        print_grades(grades)

def print_grades(grades):
    for i in range(len(grades)):
        print(grades[i], '\t', end=' ')
    print()
```

```
print_menu()
menu_choice = 0
while menu_choice != 6:
    print()
    menu_choice = int(input("Menu Choice (1-6): "))
    if menu_choice == 1:
        name =input("Student to add: ")
        students[name] = [0] *len(max_points)
    elif menu_choice == 2:
        name = input("Student to remove: ")
        if name in students:
            del students[name]
        else:
            print("Student:", name, "not found")
    elif menu_choice == 3:
        print_all_grades()
```

```
    elif menu_choice == 4:
        print("Record Grade")
        name =input("Student: ")
        if name in students:
            grades = students[name]
            print("Type in the number of the grade to record")
            print("Type a 0 (zero) to exit")
            for i in range(len(assignments)):
                print(i + 1, assignments[i], '\t', end=' ')
            print()
            print_grades(grades)
            which = 1234
            while which != -1:
                which = int(input("Change which Grade: "))
                which -= 1 #same as which = which - 1
                if 0 <= which < len(grades):
                    grade =int(input("Grade: "))
                    grades[which] = grade
                elif which != -1:
                    print("Invalid Grade Number")
        else:
            print("Student not found")
    elif menu_choice != 6:
        print_menu()
```

```
1. Add student
2. Remove student
3. Print grades
4. Record grade
5. Print Menu
6. Exit

Menu Choice (1-6): 3
       hw ch 1          hw ch 2          quiz             hw ch 3          test
#Max    25               25               50               25               100

Menu Choice (1-6): 5
1. Add student
2. Remove student
3. Print grades
4. Record grade
5. Print Menu
6. Exit

Menu Choice (1-6): 1
Student to add: Bill

Menu Choice (1-6): 4
Record Grade
Student: Bill
Type in the number of the grade to record
Type a 0 (zero) to exit
1   hw ch 1     2   hw ch 2     3   quiz        4   hw ch 3     5   test
0               0               0               0               0
```

```
Change which Grade: 1
Grade: 25
Change which Grade: 2
Grade: 24
Change which Grade: 3
Grade: 45
Change which Grade: 4
Grade: 23
Change which Grade: 5
Grade: 95
Change which Grade: 0

Menu Choice (1-6): 3
       hw ch 1          hw ch 2          quiz             hw ch 3          test
#Max    25               25               50               25               100
Bill    25               24               45               23               95

Menu Choice (1-6): 6
```

# End