Dictionaries: A powerful Structured Types

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Dictionary

- Dictionaries are a very powerful structure - very commonly used
- Dictionaries are used to store data items in key:value pairs
- keys in a dict must be unique, i.e. no duplicates (keys form a set)
- Values in items can be of any data type, and can be changed
- Dictionary items referred to by using the key name (no index)
- Dictionary is mutable but the keys must be of an immutable type.
- Duplicate items (i.e. with same key value) not permitted
- A dictionary can be created using the following syntax :

```
car = { "make": "Honda", "year": 2021,
        "model": "City","cc":1500, "price": 19.5,}
```

Accessing Dictionary Items

- Can access any value using key and can also assign a new value. Eg: car["make"] is "Honda", car["cc"] is 1500, car["price"] is 19.5
- Can use get method also, e.g. car.get("model") will return "City"
- If we try to access a key value which is not present - KeyError.
- However, get() returns None, if key not present; we can also specify a value that is returned when the key is absent

  car.get(“Fuel”) # returns None

  car.get(“Fuel”,”Petrol”) # returns Petrol

Keys, Values, Items

- Can get all the keys or values of a dictionary as a list

  k = car.keys() # returns all the keys

  vals = car.values() # returns all the values
- Can also get all items - this will be a list of tuples

  car.items() # returns list of type: (key, value)
- Can check whether a key exists in dictionary using the **in** keyword.

  “make” in car # Returns True

  “fuel” in car # Returns False
- **not in** for checking absence of a key

  “fuel” not in car # Returns True
- Number of key-value pairs can be obtained using the *len()* function.

Modifying Dictionary

- Cannot change a key (immutable)
- Change an item's value - just access it and assign new value

  car["make"] = "Suzuki"

- Adding an item - like change, if key doesn't exist, new item created

  car["boot"] = 450 # will add this item ("boot": 450)
- Removing an item - pop("key") will remove the item

  car.pop("make") # removes the make item, returns the value

  popitem() # removes the last item added, returns item

  clear() # clears the dictionary

  del removes the specified key from the dictionary.

Looping Over a Dictionary

```
for i in car.keys(): # looping over key values
    print(i)  # print the ith key
    print(car[i]) # print the value of ith item
for i in car(): # also loops over key values

for i in car.values(): # looping over values
    print(val) # prints the ith item value

for i in car.items(): # each i is a tuple giving the ith item
    print(i)

for x,y in car.items():  # get the key in x and value in y
    print(x,y)
```

Quiz – Single Correct

What would be the output of the code given below?

```
p = {5 : 5, 7 : '7', '5' : 5, '7' : 8}
p[7] = 5
p['5'] = 7

print(p[str(p[p[str(p[5])]])])
```

a.) Error

b.) 5

c.) 7

d.) ‘7’

Quiz – Single Correct

What would be the output of the code given below?

```
p = {5 : 5, 7 : '7', '5' : 5, '7' : 8}
p[7] = 5
p['5'] = 7

print(p[str(p[p[str(p[5])]])])
```

a.) Error

b.) 5

c.) 7

d.) '7'

Explanation : After update, p = {5: 5, '5': 7, 7: 5, '7': 8}

p[5] = 5

p[str(p[5]) = p('5') = 7

p[p[str(p[5])]] = p[7] = 5

p[str(p[p[str(p[5])]])] = p['5'] = 7 (Ans.)

Quiz

```
rate = {'jeans': 90, 'shirt': 80, 'coat': 120, 'shoes': 100, 'tie': 50}
expensive_products = [x for x,v in rate._____ if v>90]
print(expensive_products)
```

Suppose in a clothing shop, any product above the price of 90 is considered to be expensive by you. Consider the above mentioned code, what should be filled in the blank (in red) so that code outputs the following : `['coat', 'shoes']`

Note: Do not use any spaces (' ') in your answer.

Solution

```
rate = {'jeans': 90, 'shirt': 80, 'coat': 120, 'shoes': 100, 'tie': 50}
expensive_products = [x for x,v in rate.items() if v>90]
print(expensive_products)
```

Suppose in a clothing shop, any product above the price of 90 is considered to be expensive by you. Consider the above mentioned code, what should be filled in the blank (in red) so that code outputs the following : `['coat', 'shoes']`

Solution: items()

Create, Copy a Dictionary

- Like lists, assigning the dictionary variable to another does not make another copy of the dictionary
- If d1 is a dictionary, d2 = d1 means that d2 points to the same dictionary as d1 - making change in one will be reflected in other.
- To make a copy, like in list, use copy method
- d2 = d1.copy() # now d2 points to a different object
- The function dict(d1) can also be used; dict() creates empty dict
- A new dictionary can be created from a list of keys, with a default value for each item

  dict().fromkeys(<keylist>, value) #can be on any dict obj

Nested Dictionary

- The value in each item can be a dictionary (or any structured type)
- With values being dictionaries, we have nested dictionary
- Nesting can be arbitrarily deep - giving power to represent a wide range of data
- Nested dictionaries used widely through JSON format for exchanging data
- Allow a range of data structures to be captured represented as dictionaries

Example of Nested Dictionary

- Let's take the record of a person at IIIT-D.

```
p1 = {"rollno":"iiitd123", "name": "Ayush",
      "DOB": {"date": 4, "month": 2, "year": 2001},
      "Address": {"Colony": "Vasant Kunj", "No": 201},
           "Hobbies":    ["reading",    "movies",    "cricket"]
                                                              }
```

- p1["name"] is "Ayush"
- p1[“DOB”] is {"date": 4, "month": 2, "year": 2001}
- p1["DOB"]["year"] is 2001

Example of Nested Dictionary

If you want to access student records from roll no, then we want a dictionary with roll no as keys

```
    stu = {
        "rollno1":{"name": "Ayush",
          "DOB": {"date": 4, "month": 2, "year": 2001},
          "Address": {"Colony": "Vasant Kunj", "No": 201},
                "Hobbies":    ["reading",    "movies",    "cricket"]
          },
        "rollno2": {"name": "Pratush",
          "DOB": {"date": 14, "month": 12, "year": 2002},
          "Address": {"Colony": "IIITD", "No": 201},
                "Hobbies":    ["tennis",    "gaming",    "travel"]
          }
    }
stu["rollno1"]["name"] is "Ayush"
stu["rollno1"]["Address"]["Colony"] is "Vasant Kunj"
```

Dictionary Methods

- Various methods of a dictionary, which we have already seen

  d.get(<key>) # returns value of item with <key>;

  d.get(<key>, value) # returns value if key does not exist

  d.keys() # list of keys

  d.values() # list of values

  d.items() # list of types of (key, value)

  d.pop(<key>) # removes the item with <key>; removes the last item added, if key not provided

  d.clear() # clears

  d.copy() # copies the dictionary

  len(d) # size of dictionary i.e. number of keys

Example

Count frequency of list elements

Input: [1,2,1,1,2,4,6,4,1,7]

Output:

1: 4

2: 2

4: 2

6: 1

7: 1

Approach:

Idea: Create a dict of no:count

Create an empty dict d

Loop over the input

- If item not a key in d, add d:1 as an item
- If item there, increment the value of count

Print the dict d

Example...

```
# Direct approach

lst = [1,2,1,1,2,4,6,4,1,7]
freq = {} # initializing dict
for x in lst:
  if x not in freq:
    freq[x] = 0
  freq[x] = freq[x]+1
print(freq)
```

```
# Alt Method: using get()
# get(key, val) returns val if
key does not exist

lst = [1,2,1,1,2,4,6,4,1,7]
freq = {}
for x in lst:
  freq[x] = freq.get(x,0)+1
print(freq)
```

Example

Consider a student record in a college over different semesters

```
student = {
   "rollno": "1234",
   "name": "Shyam",
   "sem1": [("m101", 4, 9), ("cs101", 4, 8), ("com101", 4, 10)],
   "sem2": [("m102", 4, 8), ("cs102", 4, 9), ("ee102", 4, 8)],
   "sem3": [("m202", 2, 10), ("cs201", 4, 8), ("elect1", 4, 10)],
   }
```

Compute the SGPA for a semester; extend to compute sgpa for all semesters (remember a student may have diff no of semesters)

```
student = {
   "rollno": "1234",
   "name": "Shyam",
   "sem1": [("m101", 4, 9), ("cs101", 4, 8), ..],
   "sem2": [("m102", 4, 8), ("cs102", 4, 9), ..],
   "sem3": [("m202", 2, 10), ("cs201", 4, 8),..],
   }
```

A sem record has key of type "semn" and the value is a list of tuples (name, credits, grade), and student[sem] is that list

Number of semesters may be different

Approach (for a sem):

Fn to compute sgpa for a sem needs as input: student dict, and semester i.e. def sgpa(stu,sem)

For sem, get the list of tuples

Loop over the items, to find total credits (TC) and credits*grade

After the loop divide tot c*g by tot credits

Approach (for all sems):

Loop over all items, if key is "semn"then, call above fn - passing it the student record, and the key (as sem)

print the sgpa for this sem

Example...

```
def sgpa(student, sem):
    tot1 = 0
    tot2 = 0
    for i in student[sem]:
        tot1 += i[1]
        tot2 += i[1]*i[2]
    SGPA = tot2/tot1
    return round(SGPA, 2)

SGPA = sgpa(student, "sem1")
print(f'sgpa is {SGPA}')
```

```
#Computing SGPA for all the semesters

for i in student.items():  # i is a tuple
    if i[0][:3] == "sem":
        SGPA = sgpa(student, i[0])
        print(f'sgpa for {i[0]} is {SGPA}')

for i in student.keys():  # or just i in student
    if i[:3] == "sem":
        SGPA = sgpa(student, i)
        print(f'sgpa for {i} is {SGPA}')
```

Summary – Dictionaries

- Dictionaries store data in key-value pairs; keys or values can be of any type
- Items in dictionary are accessed using the key as "index"
- Dictionaries are mutable, but Keys cannot change, and keys must be unique
- keys(), values(), items() - provide as lists of items/tuple
- Can add an item (by just dict[key] = value), delete by pop()
- Many methods on dictionaries
- Can loop over dictionary - using keys. Can also loop over values by getting values(), or items()

Quiz

Consider the following code to compute class average for an exam of the “Digital Circuits” course.

Fill in the blanks with appropriate code to perform the same.

Hint: use sum() function

Note: Do not use any spaces (' ') in your answer.

```
course = {"title": "Digital Circuits",
          "instructor": "Gustavo",
          "days": ['Mon', 'Wed', 'Fri'],
          "statistics": {
              "attendance": [75, 50, 90, 25, 80],
              "scores": {
                  "labs": [7, 5, 10, 3, 8],
                  "exam": [8, 4, 6, 2, 10],
                  "assignment": [70, 65, 40, 80, 100]
              },
              "num_students": 5
          }
        }
exam_class_average = _______ / course['stats']['num_students']
print(exam_class_average)
```

Solution

Consider the following code to compute class average for an exam of the “Digital Circuits” course.

Fill in the blanks with appropriate code to perform the same.

Hint: use sum() function

Note: Do not use any spaces (' ') in your answer.

```
sum(course['stats']['scores']['exam'])
```

```
course = {"title": "Digital Circuits",
          "instructor": "Gustavo",
          "days": ['Mon', 'Wed', 'Fri'],
          "stats": {
              "attendance": [75, 50, 90, 25, 80],
              "scores": {
                  "labs": [7, 5, 10, 3, 8],
                  "exam": [8, 4, 6, 2, 10],
                  "assignment": [70, 65, 40, 80, 100]
              },
              "num_students": 5
          }
         }

exam_class_average = sum(course['stats']['scores']['exam']) /
course['stats']['num_students']

print(exam_class_average)
```

Structured Types
Some Commonalities

Creating Objects using Constructor

- Python provides constructor functions to create an object of a type from other objects,

  list(), set(), dict(), tuple(), str() # the arg has to be suitable

- Example uses of these - valid and invalid

  T = tuple("Hello") # ('H', 'e', 'l', 'l', 'o')

  S = str(5.0) # Converted number to string. Now S = ‘5.0’

  L1 = list(T) # ['H', 'e', 'l', 'l', 'o']

  L2 = list(“Hello”)

  D = dict(a=1,b=2) # {'a': 1, 'b': 2}

Common Functions on Objects

There are some common functions which are useful - some apply to structured types, some to all objects

- type(x) # returns the type of x
- len(x) # returns the no of items in x
- all(x) # returns True if all elements of an iterable are true
    - all([1,1,1]) -> True ; all([1,0,0]) -> False
- any(x) # returns True if any elements of an iterable are true
    - any([1,0,0]) -> True ; any([0,0,0]) -> False
- reversed(x) # returns a reverse iterator
- sorted(x) # returns a new sorted list from the items in iterable

enumerate() - convenient way to iterate

This is a special function that allows you to simultaneously get the indexes and values of an iterable object.

Examples:

```
l = ["Sam", "John", "Tim"]
for index, val in enumerate(l):
    print(index, val)
```

```
0 Sam
1 John
2 Tim
```

```
d = {"Sam":10, "John":30, "Tim":20}
for index, (key, val) in enumerate(d.items()):
    print(index, key, val)
```

```
0 Sam 10
1 John 30
2 Tim 20
```

Mutable and Immutable Objects

- A variable (in python) points to an object with a value/state
- Objects are of two types: Mutable and Immutable
- Mutable objects: The state/value can be changed
- Immutable objects - state cannot be changed (though new objects can be created)
- Immutable Objects : objects of type: int, float, bool, string
- Mutable Objects: of type list, dictionary, set, (and custom objects)
- Tuple - a special case. While tuple is immutable, it may have elements (e.g. list) which are mutable

Mutable and Immutable Objects

Immutable objects - assigning a var pointing to one to another

X = 10

Y = X

Y = 20 # a new object 20 is created to which y points, x continues to point at object with value 10

Mutable objects - assigning only creates a pointer

L1 = [1, 3, 5, 9]

L2 = L1

l1 [2] = 8 # this changes the list, and so l2 will also reflect it

Quiz

Which of the following statements are correct ? (MSQ)

(A) Mutable objects in python can only be modified by adding new elements or deleting elements from them.

(B) Some immutable objects in python can contain elements that are mutable objects which can be modified.

(C) Given a dictionary ‘d’ and x=list(d.items()), both ‘d’ and ‘x’ are mutable objects.

(D) None of the above

Quiz

Which of the following statements are correct ?

(A) Mutable objects in python can only be modified by adding new elements or deleting elements from them.

(B) Some immutable objects in python can contain elements that are mutable objects which can be modified.

(C) Given a dictionary ‘d’ and x=list(d.items()), both ‘d’ and ‘x’ are mutable objects.

(D) None of the above

Solution: (B), (D)

Extra Slides

Set Operations with Update

```
x = {1,2,3,4}
y = {2,3,5,6}

z = x.intersection(y)
print(z) # {2, 3}
print(x) # {1, 2, 3, 4}

x.intersection_update(y)
print(x) # {2, 3}
print(y) # {2, 3, 5, 6}
```

```
x = {1,2,3,4}
y = {2,3,5,6}

z = x.symmetric_difference(y)
print(z) # {1, 4, 5, 6}
print(x) # {1, 2, 3, 4}

x.symmetric_difference_update(y)
print(x) # {1, 4, 5, 6}
print(y) # {2, 3, 5, 6}
```

Changing a Tuple

- Once created a tuple cannot be changed
- One way - convert it to a list, then modify, and convert it back, eg

```
clist = list(colors)
<ops on clist> # e.g. add/remove/modify
colors = tuple(clist) # creates a new tuple
```

- Like strings some ops to create a new tuple w.r.t existing one are there.

Example

Given two lists (l1 and l2) and a sum (s) as input, find the number pairs which add up to the given sum.

Input lists:
L1 = [1,2,3,4,5]
L2 = [6,7,8,2,1]
Input sum: s = 11

Output:
[(8, 3), (7, 4), (6, 5)]

```
L1 = [1,2,3,4,5]
L2 = [6,7,8,2,1]
s = 11
res = [(s-i,i) for i in L1 if (s-i) in L2]
# Now res is [(8, 3), (7, 4), (6, 5)]
```

Extra Slides

Example

Given a list as input, create a dictionary with the factor frequency.

Input: [2,6,9,7,5,10]

Output:

{1: 6, 2: 3, 3: 2, 4: 0, 5: 2, 6: 1, 7: 1, 8: 0, 9: 1, 10:1}

```
lst = [2,6,9,7,5,10]

res = {}
for x in range(1, max(lst)+1):

    T = [1 for elmt in lst if(elmt%x)==0]
    res[x] = sum(T)

print(res)
```

Factor Frequency Computation:

1 is a factor of all 6 numbers,

2 is a factor of 3 numbers i.e. 2,6,10

3 is a factor of 2 numbers i.e. 6,9

And so on …