Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

Learning Objectives:

Tuple , Dictionaries, Dictionary Comprehension

Session 8 , 4 Oct 22

Dr N V Ganapathi Raju
Professor and HOD of IT
GRIET

© Replication or other unauthorized use of this material is prohibited

Python Tuple

- A tuple is a sequence of values , Values can be any **type**, are **indexed** by integers
- **immutable** objects; cannot be changed
- Used to create **write-protected data.**

  - are not dynamic, faster than lists

- Tuple is enclosed between parenthesis
- **Tuple Vs List**

  - Tuple is **like list**
  - Difference is, List have **mutable** objects whereas Tuple have **immutable** objects.
  - List is enclosed between square bracket, tuple between parenthesis

© Replication or other unauthorized use of this material is prohibited

Creating Tuple

- Tuples are defined in parentheses () , values are separated by commas
- Contain values of different data types.
- Can be an empty
- A single valued tuple,
    - must be a comma at the end of the value
- Can also be nested.
- If a Tuple does not enclose with parenthesis, still it will be considered as tuple

© Replication or other unauthorized use of this material is prohibited

Creating Tuple example

```
t1=(10,20.50,"python",'p',True)

t2=  10,20.50,"python",'p',True

t3 =  (10,)

t4 =  ()

t5 =  (10,20.50,"python")

t6=   (t5,'p',True)

t7 = tuple()
```

© Replication or other unauthorized use of this material is prohibited

Accessing Tuple values

- Use square bracket **[],** to slice along the **index** or **indices** and access the values of a tuple

- Tuple elements can be accessed like String and List

  - Forward Indexing , indexing start with 0 to n-1 (Reading from Left to Right)

  - Backward Indexing, indexing start with -1 to –n (Reading from Right to Left)

© Replication or other unauthorized use of this material is prohibited

Accessing Tuple elements example

t1=(10,20.50,"python",'p',True)

t1[0]

t1[4]

t1[-1]

t1[-5]

t1[5]

t1[-6]

IndexError: tuple index out of range

© Replication or other unauthorized use of this material is prohibited

Tuple Slicing

- Slicing is used to select range of values from tuple object

  syn: [start_index : end_index : step].

  - start_index is the beginning index of the slice; default value is 0.

  - end_index is the end index of the slice; default value is the len(sequence).

  - step is the amount by which the index increases, the default value is 1.

© Replication or other unauthorized use of this material is prohibited

Tuple slicing example

```
t1 = ("python","tuples","are","immutable","write","protected")

print(t1[1:4])

print(t1[:4])

print(t1[:])

print(t1[::2])

print(t1[::-1])
```

© Replication or other unauthorized use of this material is prohibited

Tuple basic operations

- **Membership operators**

  - in returns True if an item is present in sequence else False

  - not in returns True if an item is not present in sequence else False

- **Addition Tuple**

  - Tuple can be added by using the concatenation operator(+) to join two tuples.

- **Replicating Tuple:**

  - Replicating can be performed by using '*' operator by a specific number of time.

© Replication or other unauthorized use of this material is prohibited

Tuple basic operations example

```
t1 = (10,20.50,"python",'p',True)

'p' in t1

20.50 not in t1
```

```
t1 = ("python","tuples","are")

t2 = ("and","immutable","write","protected")

t3 = t1 + t2

print(t3)

t4 = ("Immutable " * 3)
print(t4)
```

© Replication or other unauthorized use of this material is prohibited

Tuple basic operations

- **Updating elements in a List:**
  - Elements of the Tuple cannot be **updated**. since Tuples are **immutable**.

- **Deleting elements from Tuple:**
  - Deleting individual element from a tuple is not supported.
  - Whole of the tuple can be deleted using the **del** statement

```
t1=(10,20,'rahul',40.6,'z')
print (t1)
del (t1)
```

© Replication or other unauthorized use of this material is prohibited

Index and Count

- index() : searches an element in a tuple and returns its index.
    - returns its position
    - if the same element is present more than once, the first position is returneda
    - If no element is found, a ValueError exception is raised indicating the element is not found.

```
t2=(10,'savik',40.6,'z')

print(t2.index(40.6))

# print(t2.index('a'))
raises     ValueError: tuple.index(x): x not in tuple
```

```
t2=(10,'savik',40.6,10,'z')

print(t2.count(10))
```

© Replication or other unauthorized use of this material is prohibited

Tuple operations

- **min(), max() and len()**

  built-in function to get the maximum value, minimum values and the length of a sequence.

```
t1=(100, 255.55, True)

print(len(t1))

print(min(t1))

print(max(t1))

print()
```

© Replication or other unauthorized use of this material is prohibited

unpacking

- **unpack** : tuple into variables.
  - when unpacking a tuple the number of variables on the

    left side should be equal to the number of the values in the tuple

    Otherwise, error such as **ValueError**: too many values to unpack

```
a, b, c = (10, 20, 30)

print(a)

print(b)

print(c)
```

© Replication or other unauthorized use of this material is prohibited

Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

Python Dictionaries

Session 8 , 4 Oct 22

Dr Ganapathi Raju

Professor and HOD of IT

GRIET

© Replication or other unauthorized use of this material is prohibited

Features of Dictionary

- **Dictionary** is an **unordered** set of **key and value pair**
- **Mutable** i.e., value can be updated.
- **Key** must be **unique** and immutable, such as numbers, strings
- V**alues** of a dictionary may **be any data type**
- **key and value** is known as **item**
- **Container** that contains data, enclosed within **curly braces**.

© Replication or other unauthorized use of this material is prohibited

Creating Dictionary

- **Dictionary** enclosed within **curly braces**.
- **The key and the value is separated by a colon (:), pair is known as item**
- **Items are separated from each other by a comma (,)**
- **Different items are enclosed within a curly brace and this forms Dictionary**

© Replication or other unauthorized use of this material is prohibited

Creating dictionaries example

```
dict1 = {'Name': 'Ajay', 'Age':30, 'Profession’ : 'Programmer'}
print(dict1)


dict2 = {}
print(type(dict2))
```

© Replication or other unauthorized use of this material is prohibited

Accessing dictionary Items

- Dictionaries value can be accessed by their keys

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession': 'Programmer'}

print(dict1)


no = dict1['ID']
print(no)


age = dict1['Age']
print(age)


name = dict1['Name']
print(name)
```

Note: if the key is not available returns Error

```
#des = dict1.['Description']

#print(des)
```

© Replication or other unauthorized use of this material is prohibited

Accessing values using get()

- **Dictionary elements also be accessed with get()**

  syn: get(“key”)

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession’: 'Programmer'}
print(dict1)

job2 = dict1.get('Profession')
print(job2)

des = dict1.get('Description')
print(des)
# Key
```

© Replication or other unauthorized use of this material is prohibited

Dictionary Mutability Updating dictionary values

- **Dictionary is mutable**
  - **new items added** or **existing items can be changed**
  - If the key is already present, value gets updated, else {key: value} pair is added to the dictionary

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession': 'Programmer'}

# update value
dict1['Name'] = "Aditya"
dict1
```

```
# add item
dict1['Description'] = "Python Programming"
dict1
```

© Replication or other unauthorized use of this material is prohibited

Updating dictionary values using update()

- **update() :** updates the dictionary with the elements from another dictionary object

  or

  from an iterable of key/value pairs.

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}
dict2 ={"Area":"Machine Learning"}
dict1.update(dict2)
print (dict1)
```

© Replication or other unauthorized use of this material is prohibited

Deleting values from dictionaries using del

- **del** statement is used for performing deletion operation

  - Item can be deleted from a dictionary using the key

    Syntax: del [key]

  - Whole dictionary can be deleted using the **del** statement

Note: For deleting specific item using Key

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession': 'Programmer'}
del dict1['ID']
dict1
```

Note: For deleting all items of dictionary

```
del dict1
```

© Replication or other unauthorized use of this material is prohibited

Deleting values from dictionaries using pop

- **pop:** removes as item with the provided key and returns the value

  - remove an item in a dictionary

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession’: 'Programmer'}

dict1.pop ('ID')

dict1
```

© Replication or other unauthorized use of this material is prohibited

Deleting values from dictionaries using clear

- clear(): Remove all items form the dictionary.

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}

dict1.clear()
```

© Replication or other unauthorized use of this material is prohibited

Dictionary

- **keys() :** displays a list of all the keys in the dictionary
- **values**() : Return dictionary's values
- **Items**(): Return (key, value) in tuple pairs

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession’: 'Programmer’}

print (dict1.keys())

print (dict1.values())

print (dict1.items())
```

© Replication or other unauthorized use of this material is prohibited

Iterating dictionary elements using keys()

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}

for k in dict1.keys():

    print (k, dict1[k])
```

© Replication or other unauthorized use of this material is prohibited

Iterating dictionary elements using items()

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}
for k,v in dict1.items():
    print (k, v)
```

© Replication or other unauthorized use of this material is prohibited

Iterating dictionary elements using values()

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}

values = dict1.values()

values
```

```
for value in dict1.values():

    print(value)
```

© Replication or other unauthorized use of this material is prohibited

Dictionary len(), copy()

- **len**() : Return number of items in the dictionary

- **copy**() : Return a copy of the dictionary.

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}

print (len(dict1))
```

```
dict1 = {'ID': '100', 'Name': 'Shashank ', 'Age':30, 'Profession':'Programmer'}

dict2 = dict1.copy()

print(dict2)
```

© Replication or other unauthorized use of this material is prohibited

fromkeys()

- **fromkeys**() : creates a **new dictionary** from the given **sequence** of elements

dict.fromkeys(*keys*, *value*)

© Replication or other unauthorized use of this material is prohibited

Dictionary all(), any()

- **all**(): returns **True** if all keys of the dictionary are true

  - or if the dictionary is empty

- **any** () return **True** if any key of the dictionary is true.

  - If the dictionary is empty, returns “False”.

© Replication or other unauthorized use of this material is prohibited

Dictionary Comprehensions

Dr Ganapathi Raju
Professor and HOD of IT
GRIET

© Replication or other unauthorized use of this material is prohibited

Comprehensions

- Comprehensions are constructs that

  allow sequences to be built from other sequences.

- Python supports three kinds of comprehensions

  1. List comprehension
  2. Dictionary comprehension
  3. Set comprehension

© Replication or other unauthorized use of this material is prohibited

Adv of List comprehension

- is an **elegant way to define and create lists based on existing lists**
- **more compact and faster than normal functions and loops for** creating list

Drawback

- avoid writing very **long list comprehensions** in one line

to ensure that code is user-friendly

© Replication or other unauthorized use of this material is prohibited

Dictionary Comprehension

- Dictionary comprehensions are used when the input is in the form of a dictionary

  or a **Key : Value pair**

  dict_variable = { key: value for (key, value) in dictonary.items()}

- is a powerful concept and can be used to substitute for **loops and lambda functions**.

```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

double_dict1 = {k:v*2 for (k,v) in dict1.items()}

print(double_dict1)
```

© Replication or other unauthorized use of this material is prohibited