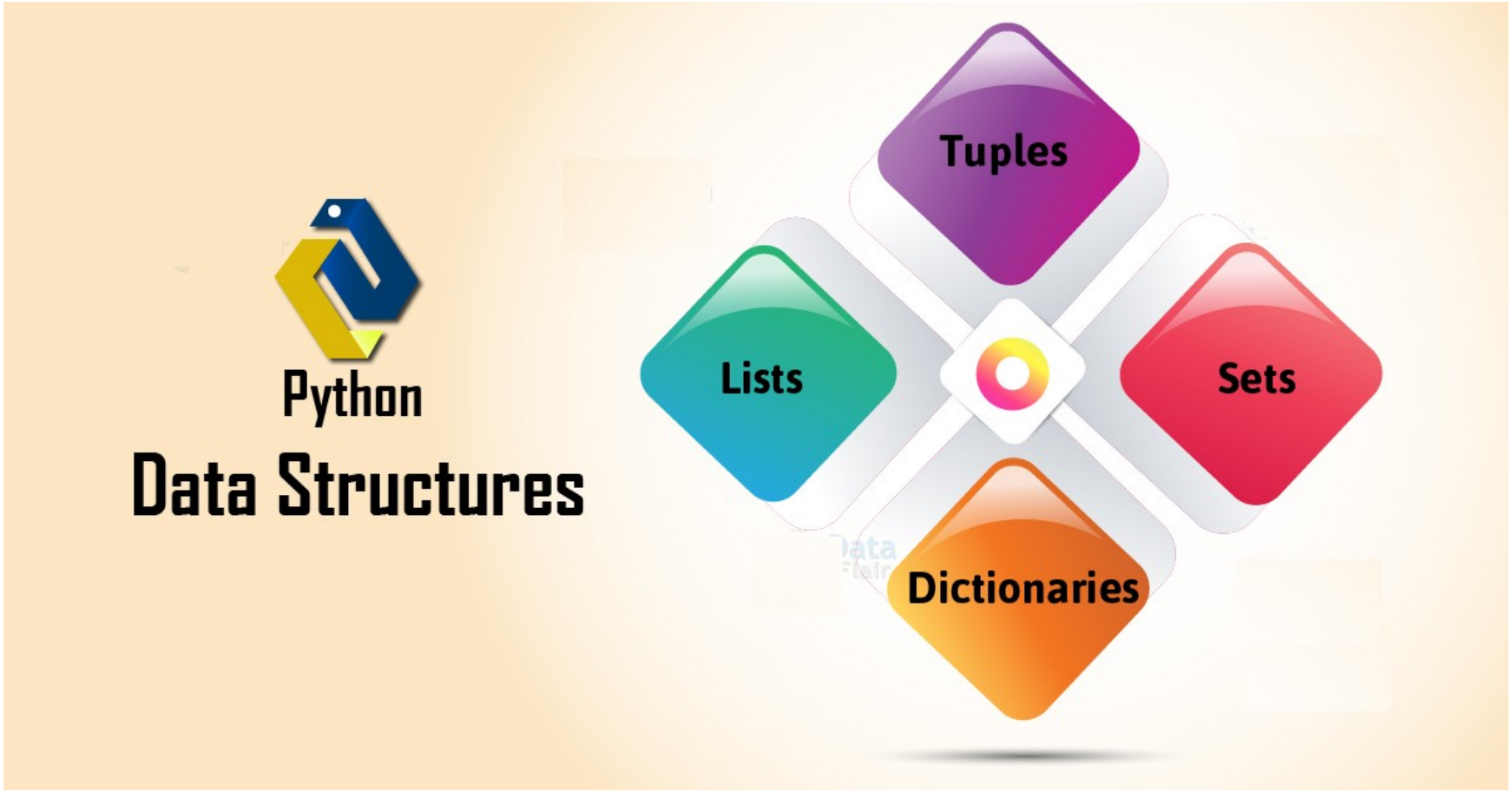
Python : Data Structures

Tushar B. Kute,
http://tusharkute.com

Data Structures



Data Structures Separators

List	[ ]
Tuple	( )
Set	{ }
Dictionary	{ : }

List

- List is a type of container in Data Structures, which is used to store multiple data at the same time.
- Lists are just like the arrays, declared in other languages.
- Lists need not be homogeneous always which makes it a most powerful tool in Python.
- A single list may contain DataTypes like Integers, Strings, as well as Objects.
- Lists are also very useful for implementing stacks and queues. Lists are mutable, and hence, they can be altered even after their creation.

List


Creating lists
Accessing lists
Slicing lists
Reassigning lists(mutable)
Deleting elements
Multidimensional Lists
Python
Lists
Concatenation of Lists
Operations on Lists
Iterating on a list
List Comprehension
Built-in Functions
Built-in Methods

Creating a list

```
>>> lang = ['Mar','Urd','San','Guj', 'Hin']
>>> type(lang)
<class 'list'>
```

- List of similar elements:

```
>>> col = ['blue','red','green','yellow']
```

- List of dissimilar elements:

```
>>> days = ['Mon','Tue','Wed',4,5,6]
```

Accessing and replacing elements

```
>>> days
['Mon', 'Tue', 'Wed', 4, 5, 6]
>>> print(lang)
['Mar', 'Urd', 'San', 'Guj', 'Hin']
```

- Indexing Starts from 0 to n-1

```
>>> print(lang[2])
San
>>> lang[1] = 'Kan'
>>> print(lang)
['Mar', 'Kan', 'San', 'Guj', 'Hin']
```

Negative Indexing

- We can access any collection element using negative indexing i.e. from second end.

```
>>> print(days)
['Mon', 'Tue', 'Wed', 4, 5, 6]
>>> print(days[-1])
6
>>> days[-2] = 'Fri'
>>> print(days)
['Mon', 'Tue', 'Wed', 4, 'Fri', 6]
```

Creating a list of list

```
>>> months = [31,[28,29],31,30,31,30]
>>> type(months)
<class 'list'>
>>> months[1]
[28, 29]
>>> months[1][0]
28
```

Slicing the list

```
>>> num = [23,67,12,48,29,38,19]
>>> num[2:6]
[12, 48, 29, 38]
>>> num[2:]
[12, 48, 29, 38, 19]
>>> num[:5]
[23, 67, 12, 48, 29]
>>> num[:]
[23, 67, 12, 48, 29, 38, 19]
```

Replacing set of elements

```
>>> metals =['H','O','He','K','Si','Fe']
>>> metals
['H', 'O', 'He', 'K', 'Si', 'Fe']
>>> metals[2:4] = ['Na','S']
>>> metals
['H', 'O', 'Na', 'S', 'Si', 'Fe']
```

Existence of element using ‘in’

```
>>> metals = ['H','O','He','K','Si']
>>> 'O' in metals
True
>>> 'he' in metals
False
>>> 'O' not in metals
False
```

Deleting elements

```
>>> metals = ['H','O','He','K','Si']
>>> del metals[1]
>>> metals
['H', 'He', 'K', 'Si']
>>> del metals[1:3]
>>> metals
['H', 'Si']
>>> del metals
>>> metals
NameError: name 'metals' is not defined
```

Multi-dimensional list

```
>>> mat = [[1,2,3],[5,6,7],[4,[9,8]]]
>>> mat
[[1, 2, 3], [5, 6, 7], [4, [9, 8]]]
>>> mat[2]
[4, [9, 8]]
>>> mat[2][1]
[9, 8]
>>> mat[2][1][0]
9
```

Arithmetic operations

```
>>> num = [7,2,5,3,6]
>>> num * 2
[7, 2, 5, 3, 6, 7, 2, 5, 3, 6]
>>> num + [4,8]
[7, 2, 5, 3, 6, 4, 8]
>>> num += [0,9]
>>> num
[7, 2, 5, 3, 6, 0, 9]
```

Iterating through list

```
>>> num = [7,2,5,3,6]
>>> for n in num:
        if n % 2 != 0:
            print(n, end=' ')

7 5 3
```

List comprehension

- You can create a new list just like you would do in mathematics. To do so, type an expression followed by a for statement, all inside square brackets.
- You may assign it to a variable. Let's make a list for all even numbers from 1 to 20.

```
>>> even = [2*i for i in range(1,11)]
>>> even
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Empty list

- You can create a list without any element in Python.

```
>>> num = []
>>> num
[]
>>> type(num)
<class 'list'>
>>> len(num)
0
```

Assigning list elements

```
>>> num = [1,2,3]
>>> x,y,z = num
>>> x
1
>>> y
2
>>> z
3
```

Built-in functions

Built-in functions – 1

- len( )
  - It calculates the length of the list.
- max( )
  - It returns the item from the list with the highest value.
- min( )
  - It returns the item from the Python list with the lowest value.
- sum( )
  - It returns the sum of all the elements in the list.

Example:

```
>>> num = [7,2,5,8,6,4,3]
>>> len(num)
7
>>> max(num)
8
>>> min(num)
2
>>> sum(num)
35
```

Built-in functions – 2

- sorted( )
  - It returns a sorted version of the list, but does not change the original one.
- list( )
  - It converts a different data type into a list.
- any( )
  - It returns True if even one item in the Python list has a True value.
- all( )
  - It returns True if all items in the list have a True value.

Sort

```
>>> num = [7,2,5,8,6,4,3]
>>> sorted(num)
[2, 3, 4, 5, 6, 7, 8]
>>> num = [7,2,'X',5,8,True]
>>> sorted(num)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: str() < int()
```

list( ) and any( )

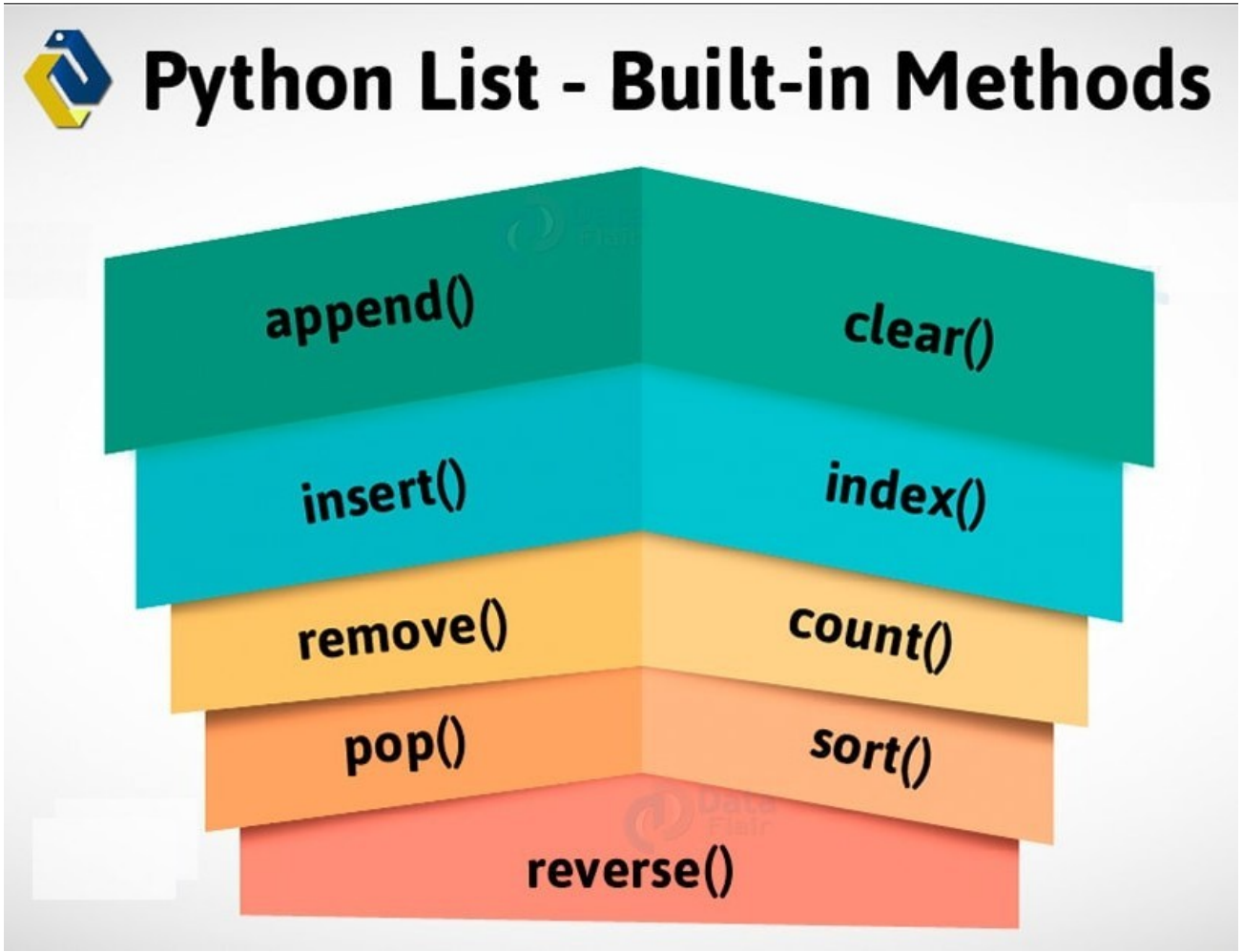
```
>>> list('xyz')
['x', 'y', 'z']
>>> data = [0,0,0,0,False]
>>> any(data)
False
>>> data = [0,0,3,0,False]
>>> any(data)
True
```

tusharkute.com

all()

```
>>> data = [0,0,3,0,False]
>>> all(data)
False
>>> data = [4,7,2,4,6,7]
>>> all(data)
True
>>> data = [4,7,2,0,6,7]
>>> all(data)
False
```

Built-in methods



Built-in methods

```
>>> num = [7,2,5,8,6,4,3]
>>> num.append(0)
>>> num
[7, 2, 5, 8, 6, 4, 3, 0]
>>> num.insert(1,5)
>>> num
[7, 5, 2, 5, 8, 6, 4, 3, 0]
>>> num.remove(5)
>>> num
[7, 2, 5, 8, 6, 4, 3, 0]
```

Built-in methods

```
>>> num = [7,2,5,8,6,4,3]
>>> x = num.pop()
>>> x
3
>>> num
[7, 2, 5, 8, 6, 4]
>>> num.index(8)
3
>>> num.count(3)
0
```

Built-in methods

```
>>> num = [7,2,5,8,6,4,3]
>>> num.sort()
>>> num
[2, 3, 4, 5, 6, 7, 8]
>>> num.reverse()
>>> num
[8, 7, 6, 5, 4, 3, 2]
```

Reading and using elements

```
arr = []
n = int(input('How many elements? '))
for x in range(n):
    num = int(input('Enter number:'))
    arr.append(num)

print('List is:', arr)
```

Exercise

- This is a list of some metallic elements.

```
metals = [ 'silver','gold', ... ]
```

- Make a new list that is almost identical to the metals list: the new contains the same items, in the same order, except that it does NOT contain the item 'copper'.

Solution

```
metals = ['gold','copper','silver',
          'iron','aluminium','copper']
new_metals = []
for metal in metals:
    if metal != 'copper':
        new_metals.append(metal)

print(new_metals)
```

Exercise

- Given the list of ten elements. Sort only higher order 50% list and print it.

  Ex. Before: [5, 6, 2, 8, 9, 1, 3, 7, 4, 0]

  After: [5, 6, 2, 8, 9, 0, 1, 3, 4, 7]
- Given the list of ten elements. Sort odd elements so even elements will not change their positions.

  Before: [12, 13, 6, 10, 3, 1, 17, 18, 14, 15]

  After: [12, 1, 6, 10, 3, 13, 15, 18, 14, 17]

Exercise

- Sort ratio is the percentage of elements present in the ascending or descending order in a list. For example: List: [12, 13, 6, 10, 3, 1, 17, 18, 14, 15] has sort ratio: 55.56. Write a program to read list of 10 elements and find ascending sort ratio.
- Write a program to count a total number of duplicate elements in an array. Ex. List – [3,6,4,6,8,3,5,3,7,4] has total duplicate elements: 2 that is 6 and 4.

Exercise

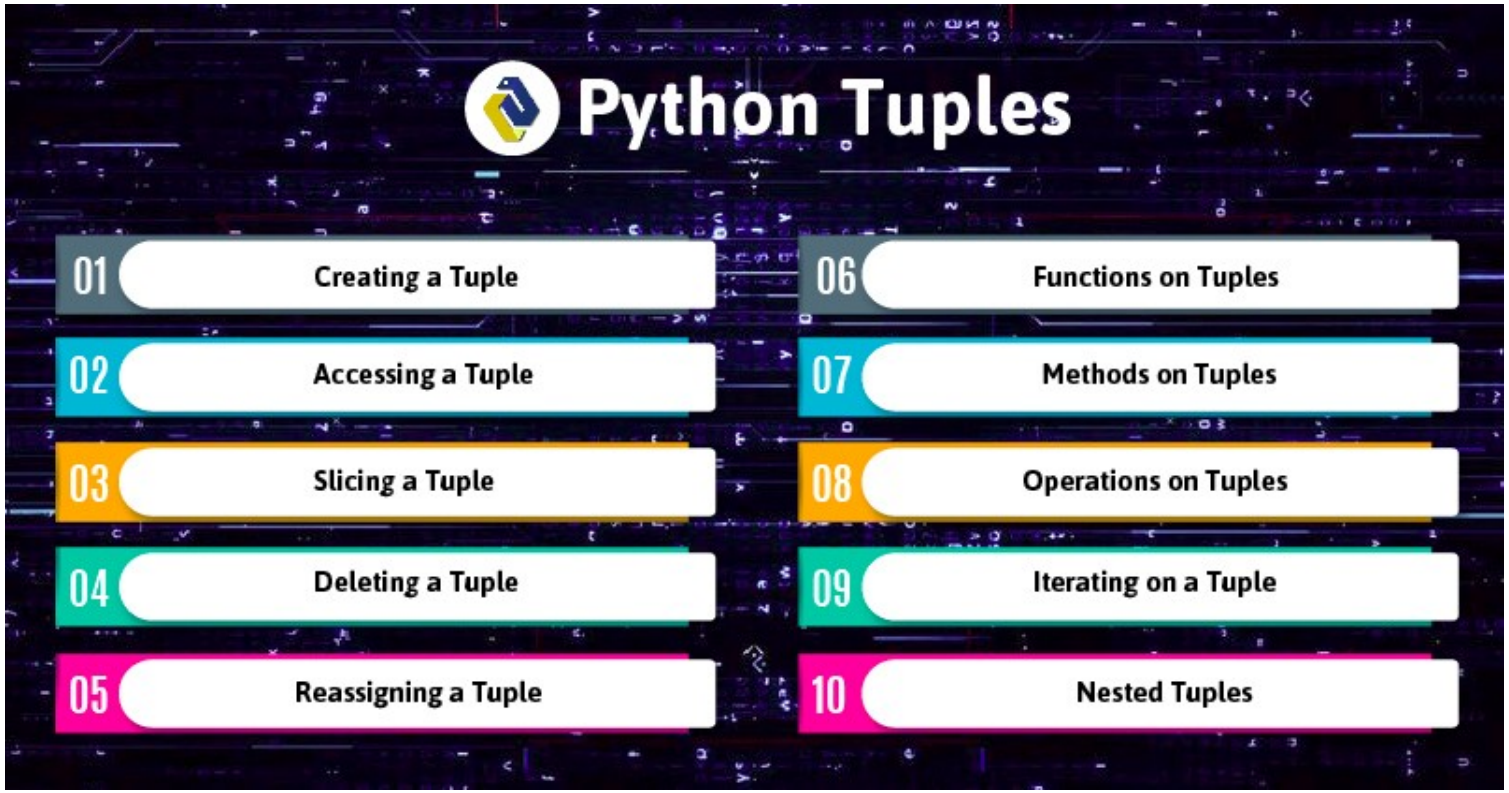
- Write a program to read 10 elements from user and create another list which contains frequency of each element from first list.
- Write a program to separate odd and even integers in separate lists.
- Write a program to find the second largest element in a list.
- Write a program for a 2D list of size 3x3 and print the matrix.
- Write a program to calculate determinant of a 3 x 3 matrix.

Exercise

- Read n number of elements from the user and find addition and average of only odd elements.
- Write a program to check whether an array is subset of another array.
  - Expected Output :
    - The given first array is : 4 8 7 11 6 9 5 0 2
    - The given second array is : 5 4 2 0 6
    - The second array is the subset of first array.

Tuple

- Python Tuples are like a list. It can hold a sequence of items. The difference is that it is immutable.



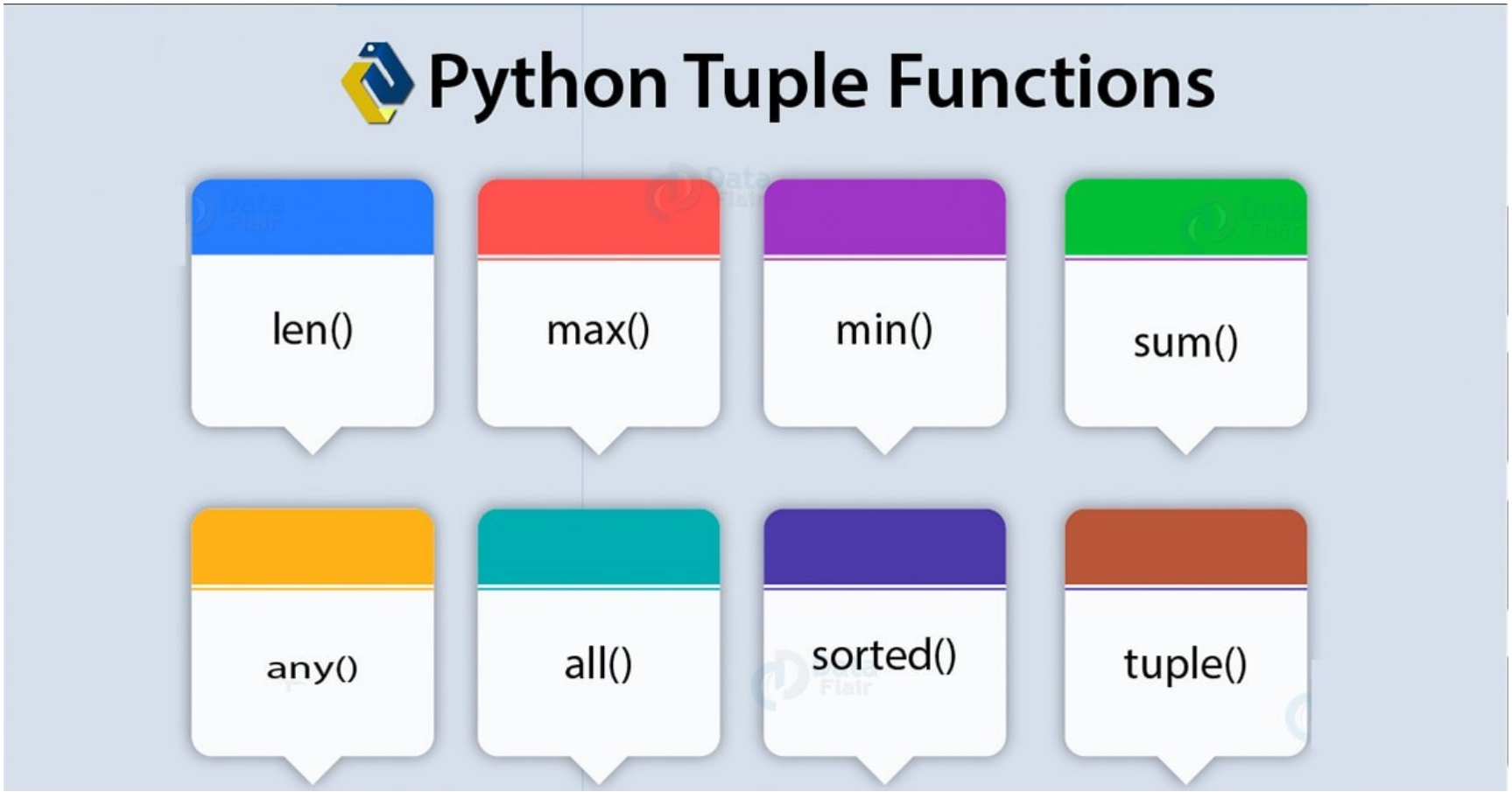
Creating a tuple

```
>>> per = (67,85,69,72,63)
>>> print(per)
(67, 85, 69, 72, 63)
>>> type(per)
<class 'tuple'>
>>> data = 1,56.34,'Ajay'
>>> data
(1, 56.34, 'Ajay')
>>> type(data)
<class 'tuple'>
```

Accessing a tuple

```
>>> per = (67,85,69,72,63)
>>> per[2]
69
>>> per[2:5]
(69, 72, 63)
>>> per[-2]
72
```

Tuple functions



Built-in functions – 1

- len( )
  - It calculates the length of the tuple.
- max( )
  - It returns the item from the tuple with the highest value.
- min( )
  - It returns the item from the tuple with the lowest value.
- sum( )
  - It returns the sum of all the elements in the tuple.

Example:

```
>>> num = (7,2,5,8,6,4,3)
>>> len(num)
7
>>> max(num)
8
>>> min(num)
2
>>> sum(num)
35
```

Built-in functions – 2

- sorted( )
  - It returns a sorted version of the tuple, but does not change the original one.
- tuple( )
  - It converts a different data type into a tuple.
- any( )
  - It returns True if even one item in the tuple has a True value.
- all( )
  - It returns True if all items in the tuple have a True value.

Sort

```
>>> num = (7,2,5,8,6,4,3)
>>> sorted(num)
[2, 3, 4, 5, 6, 7, 8]
>>> num = (7,2,'X',5,8,True)
>>> sorted(num)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: str() < int()
```

tuple( ) and any( )

```
>>> tuple('xyz')
('x', 'y', 'z')
>>> data = (0,0,0,0,False)
>>> any(data)
False
>>> data = (0,0,3,0,False)
>>> any(data)
True
```

all()

```
>>> data = (0,0,3,0,False)
>>> all(data)
False
>>> data = (4,7,2,4,6,7)
>>> all(data)
True
>>> data = (4,7,2,0,6,7)
>>> all(data)
False
```

Methods of tuple

- index()
  - This method takes one argument and returns the index of the first appearance of an item in a tuple.

    ```
    >>> per.index(63)
    ```
- count()
  - This method takes one argument and returns the number of times an item appears in the tuple.

    ```
    >>> per.count(85)
    ```

More operations

- Membership :

```
>>> 56 in per
False
>>> 81 not in per
True
```

- Concatenation:

```
>>> per + (18,43)
(67, 85, 69, 72, 63, 18, 43)
>>> new = per + (18,43)
>>> new
(67, 85, 69, 72, 63, 18, 43)
```

More operations

- Iteration:

```
>>> per
(67, 85, 69, 72, 63)
>>> for n in per:
        print(n, sep=' ')
```

- Nesting:

```
>>> num = (23,(2,7),11,36)
>>> num[1]
(2, 7)
>>> num[1][0]
2
```

Sample list of tuple

```
num = int(input("How many students?: "))
stud = []
for x in range(num):
    roll = int(input("Enter roll: "))
    name = input("Enter name: ")
    marks = float(input("Enter marks: "))
    stud.append((roll, name, marks))

for x in stud:
    print('%-2d%-10s%-3.2f' %(x[0],x[1],x[2]))
```

Example:

- Find name of the topper in the previous program.

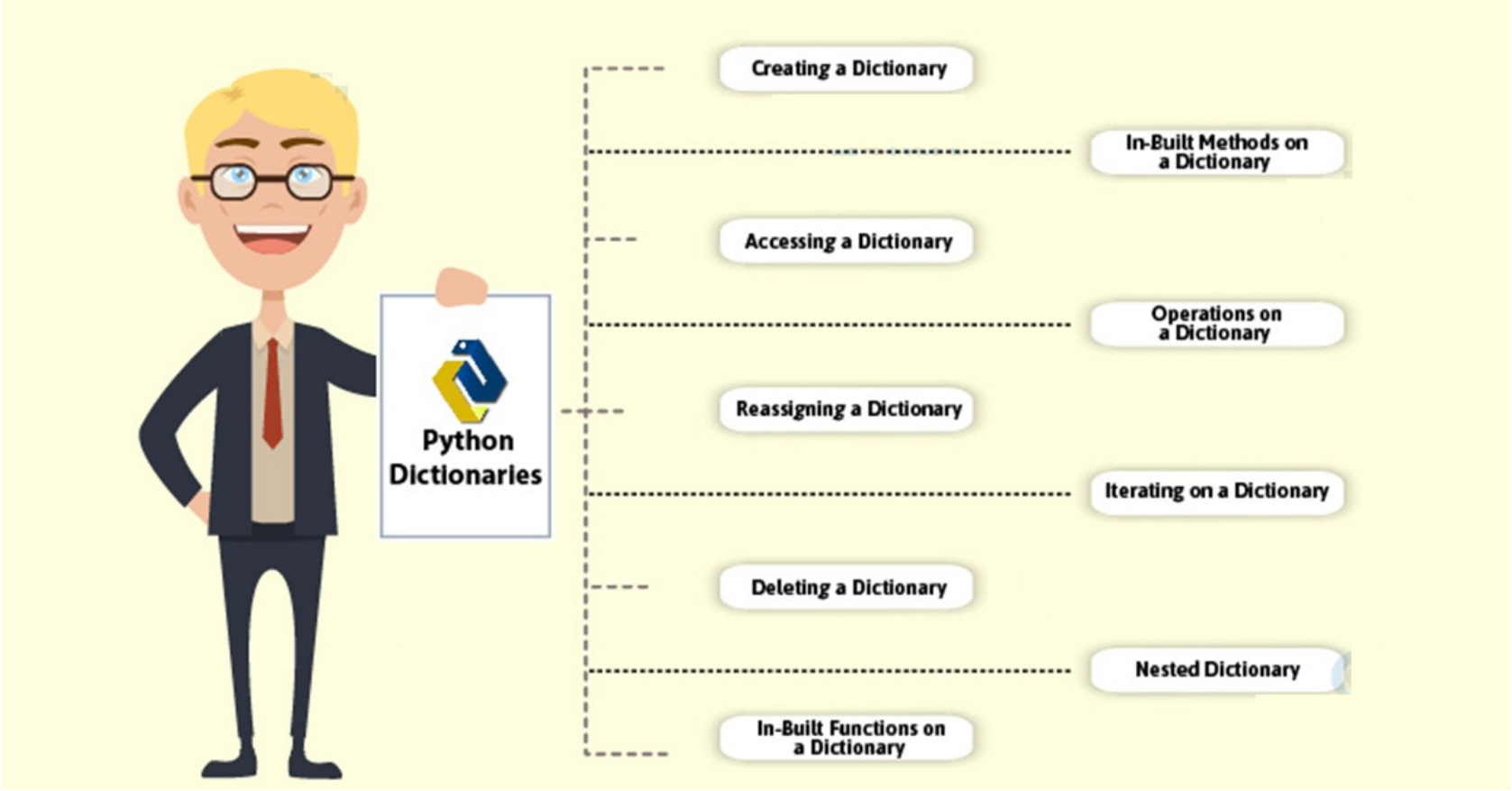
Solution

```
num = input("How many students?: ")
stud = []
for x in range(num):
    roll = int(input("Enter roll: "))
    name = input("Enter name: ")
    marks = float(input("Enter marks: "))
    t = (roll, name, marks)
    stud.append(t)
max = 0
for x in stud:
    if x[2] > max
        max = x[2]
        topper = x[1]
print(topper)
```

Dictionary

- A real-life dictionary holds words and their meanings.
- As you can imagine, likewise, a Python dictionary holds key-value pairs.

Using a dictionary



Creating a dictionary

```
>>> d = {'a':'apple','b':'ball'}
>>> num = {1:25, 3:10, 5:11}
>>> type(d)
<class 'dict'>
>>> type(num)
<class 'dict'>
>>> num = {1:25, 3:10, 5:11, 'x':'data'}
>>> type(num)
<class 'dict'>
```

Accessing and adding elements

```
>>> d['a']
'apple'
>>> num[3]
10
>>> num[2] = 20
>>> num
{1: 25, 2: 20, 3: 10, 5: 11, 'x':
'data'}
```

Using dict( ) function

```
>>> data = ([1,56],[2,49],[6,75])
>>> d = dict(data)
>>> d
{1: 56, 2: 49, 6: 75}
>>> type(d)
<class 'dict'>
```

More operations

Declaring more than once

```
>>> d = {1:45,6:23,4:23,6:27}
>>> d
{1: 45, 4: 23, 6: 27}
```

Empty dictionaries

```
>>> animals={}
>>> type(animals)
<class 'dict'>
>>> animals[1]='dog'
>>> animals[2]='cat'
>>> animals[3]='ferret'
>>> animals
{1: 'dog', 2: 'cat', 3: 'ferret'}
```

Dictionary methods

- keys( )
  - The keys() method returns a list of keys in a Python dictionary.
- values( )
  - Likewise, the values() method returns a list of values in the dictionary.
- items( )
  - This method returns a list of key-value pairs.

Dictionary methods

```
>>> data = {2:45,5:38,6:11,7:105,9:20}
>>> data.keys()
dict_keys([9, 2, 5, 6, 7])
>>> data.values()
dict_values([20, 45, 38, 11, 105])
>>> data.items()
dict_items([(9, 20), (2, 45), (5, 38),
(6, 11), (7, 105)])
```

Methods

- get()
  - It takes one to two arguments. While the first is the key to search for, the second is the value to return if the key isn't found. The default value for this second argument is None.
- clear( )
  - The clear function's purpose is obvious. It empties the Python dictionary.

Methods

```
>>> data
{9: 20, 2: 45, 5: 38, 6: 11, 7: 105}
>>> data.get(2,1)
45
>>> data.get(4,5)
5
>>> data.get(7,110)
105
```

Methods

- copy()
  - The copy() method creates a shallow copy of the Python dictionary.
- pop()
  - This method is used to remove and display an item from the dictionary. It takes one to two arguments. The first is the key to be deleted, while the second is the value that’s returned if the key isn’t found.

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

c/MITUSkillologies

Web Resources
http://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com