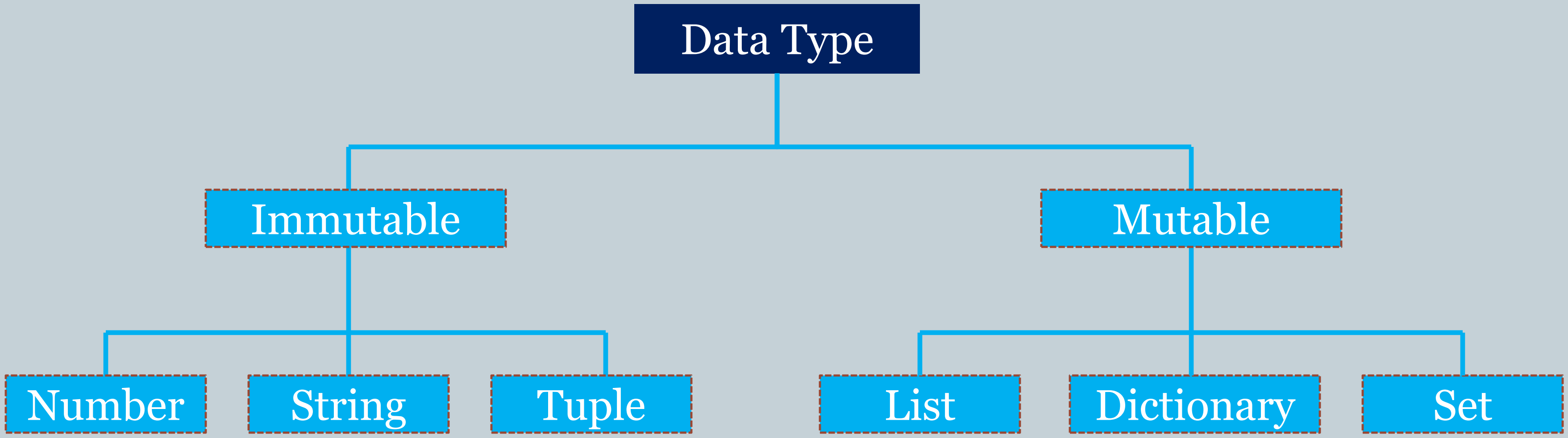
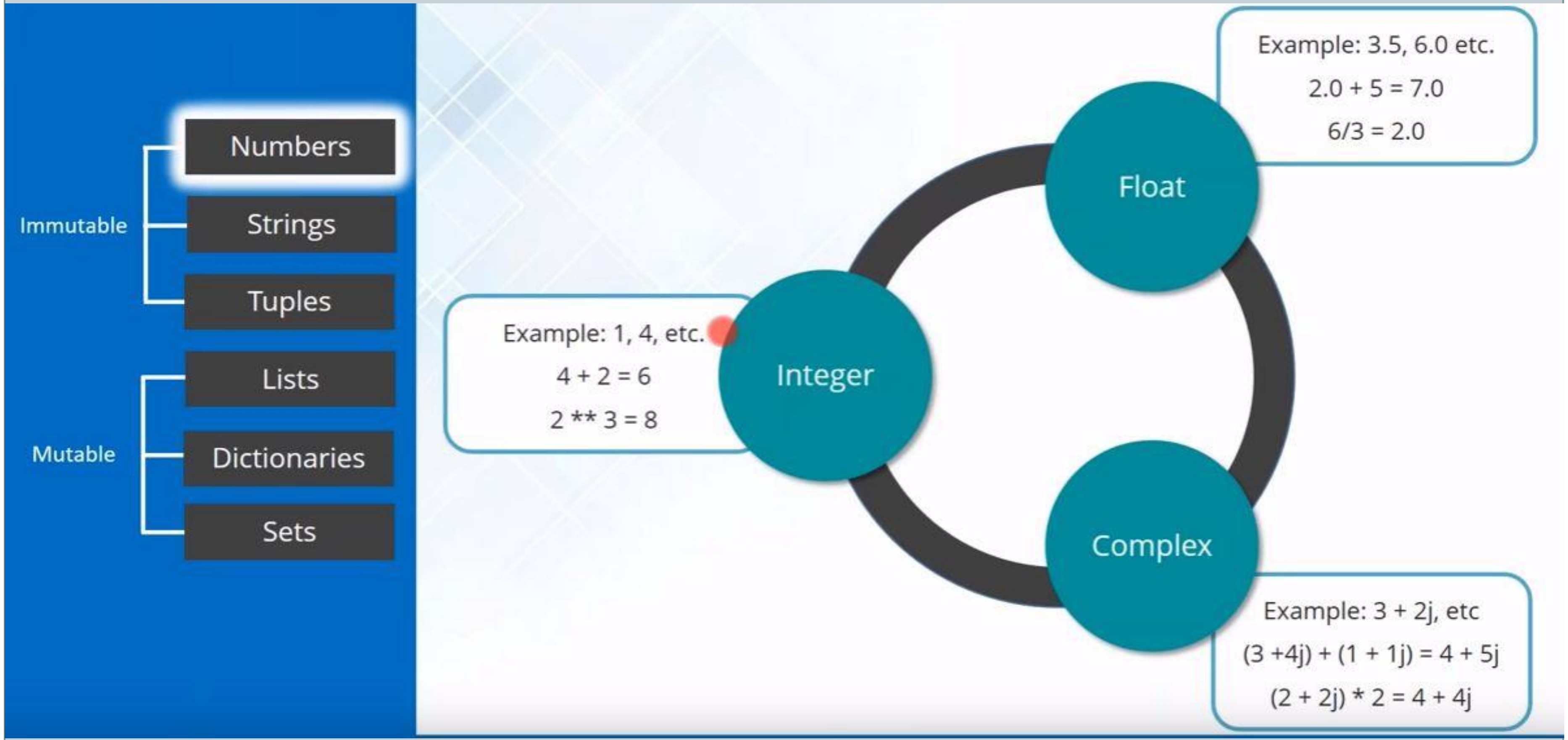
Python Data Types

Data Type



- Python is an OOP language.
- No need to define the data type of the variable before declaring a variable

Numbers

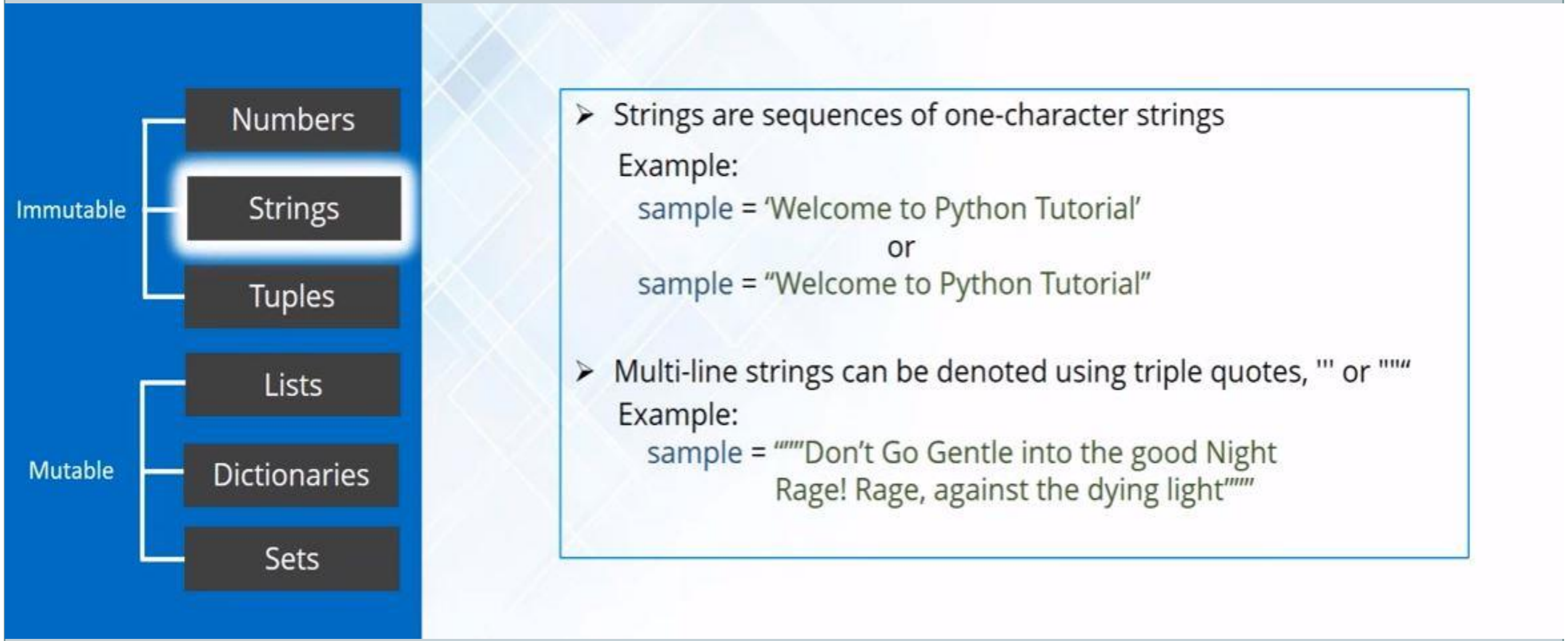


Example: 1, 4, etc.
4 + 2 = 6
2 ** 3 = 8

Example: 3.5, 6.0 etc.
2.0 + 5 = 7.0
6/3 = 2.0

Example: 3 + 2j, etc
(3 +4j) + (1 + 1j) = 4 + 5j
(2 + 2j) * 2 = 4 + 4j

String



- Strings are sequences of one-character strings

  Example:

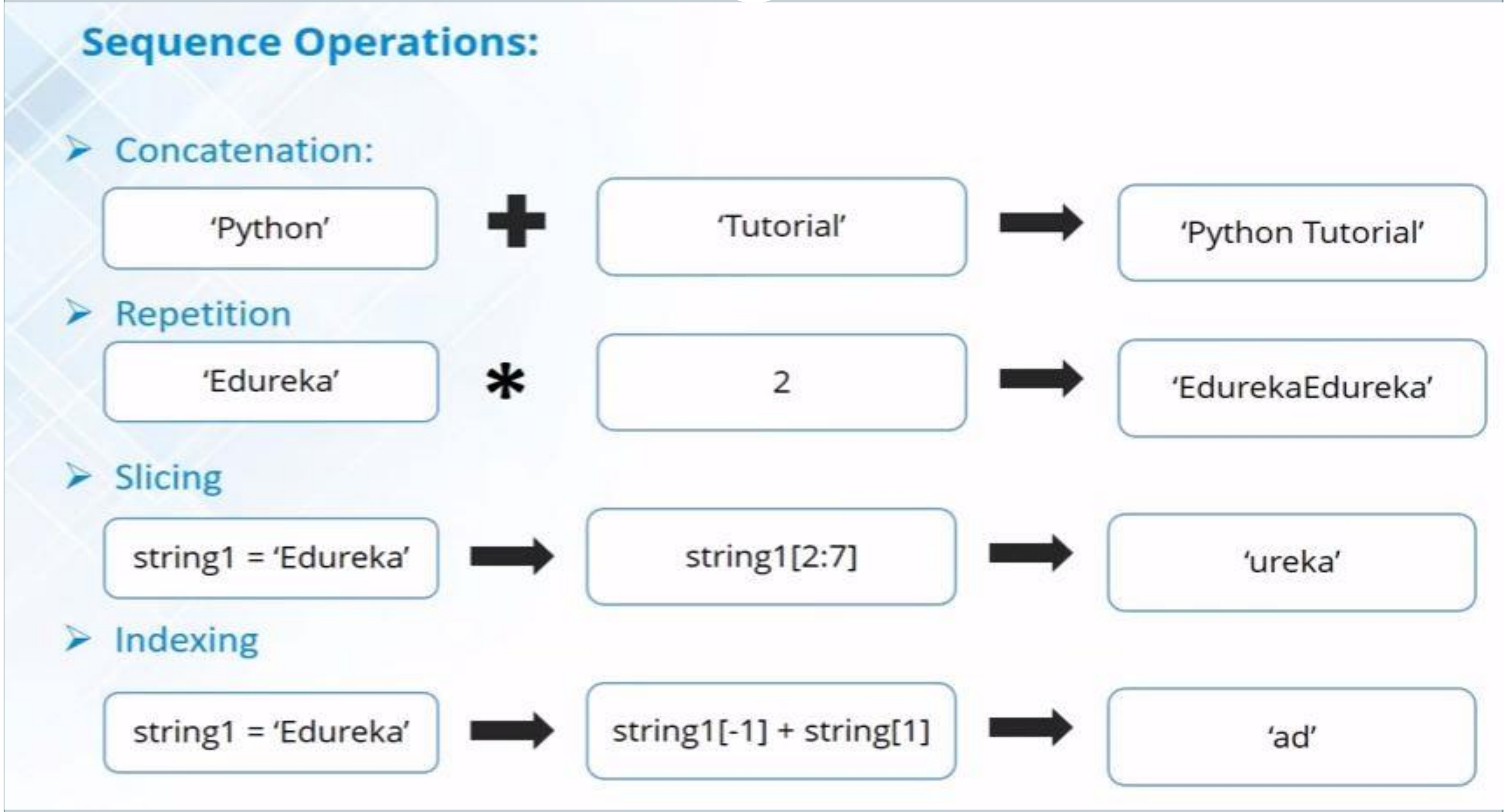
  sample = ‘Welcome to Python Tutorial’
  or
  sample = “Welcome to Python Tutorial”

- Multi-line strings can be denoted using triple quotes, ''' or """

  Example:

  sample = “””Don’t Go Gentle into the good Night
  Rage! Rage, against the dying light”””

Operation on String



Sequence Operations:

- Concatenation:

  'Python' + 'Tutorial' → 'Python Tutorial'

- Repetition

  'Edureka' * 2 → 'EdurekaEdureka'

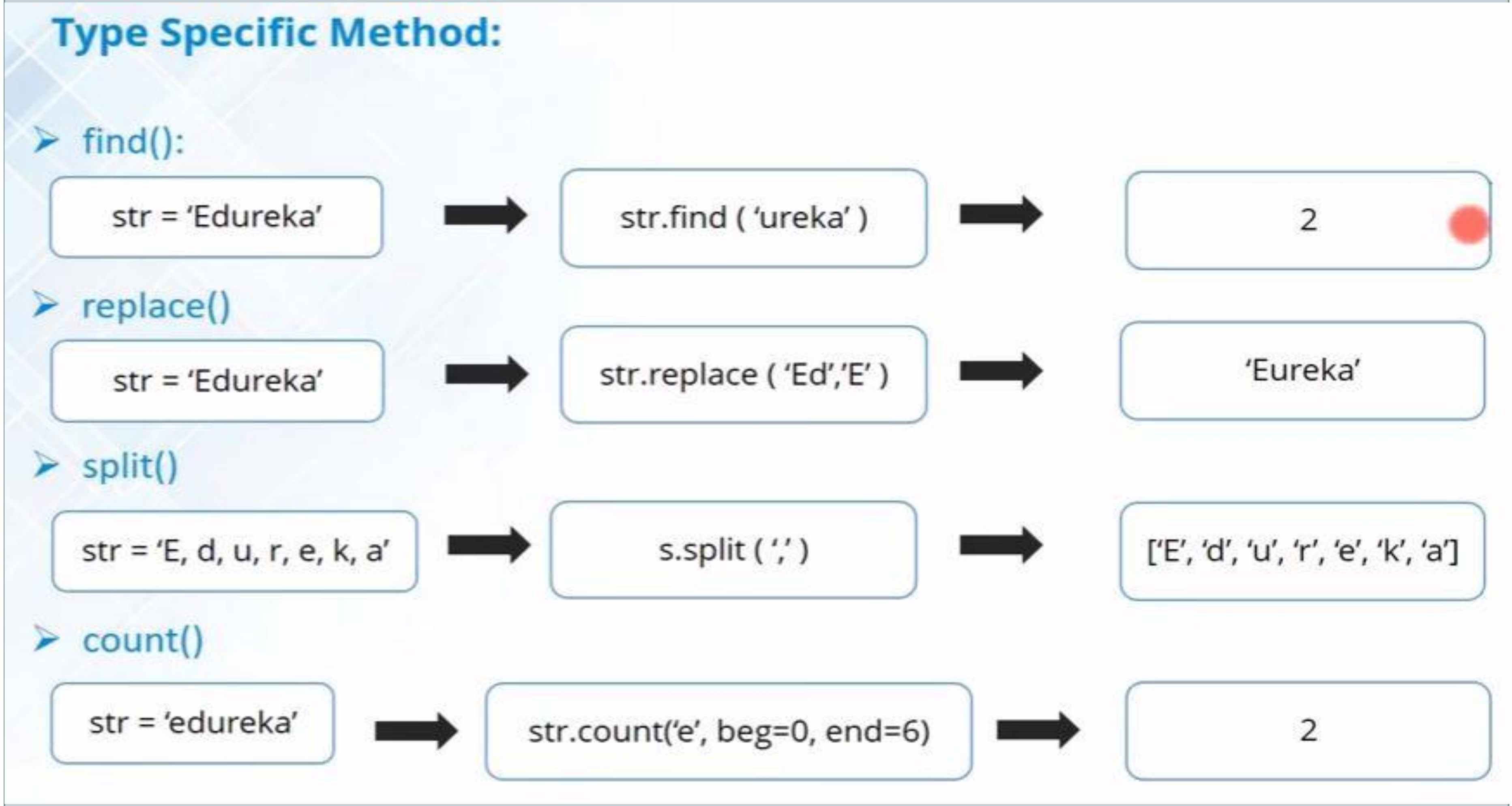
- Slicing

  string1 = 'Edureka' → string1[2:7] → 'ureka'

- Indexing

  string1 = 'Edureka' → string1[-1] + string[1] → 'ad'

Operation on String



Type Specific Method:

- find():

str = ‘Edureka’ → str.find (‘ureka’) → 2

- replace()

str = ‘Edureka’ → str.replace (‘Ed’,‘E’) → ‘Eureka’

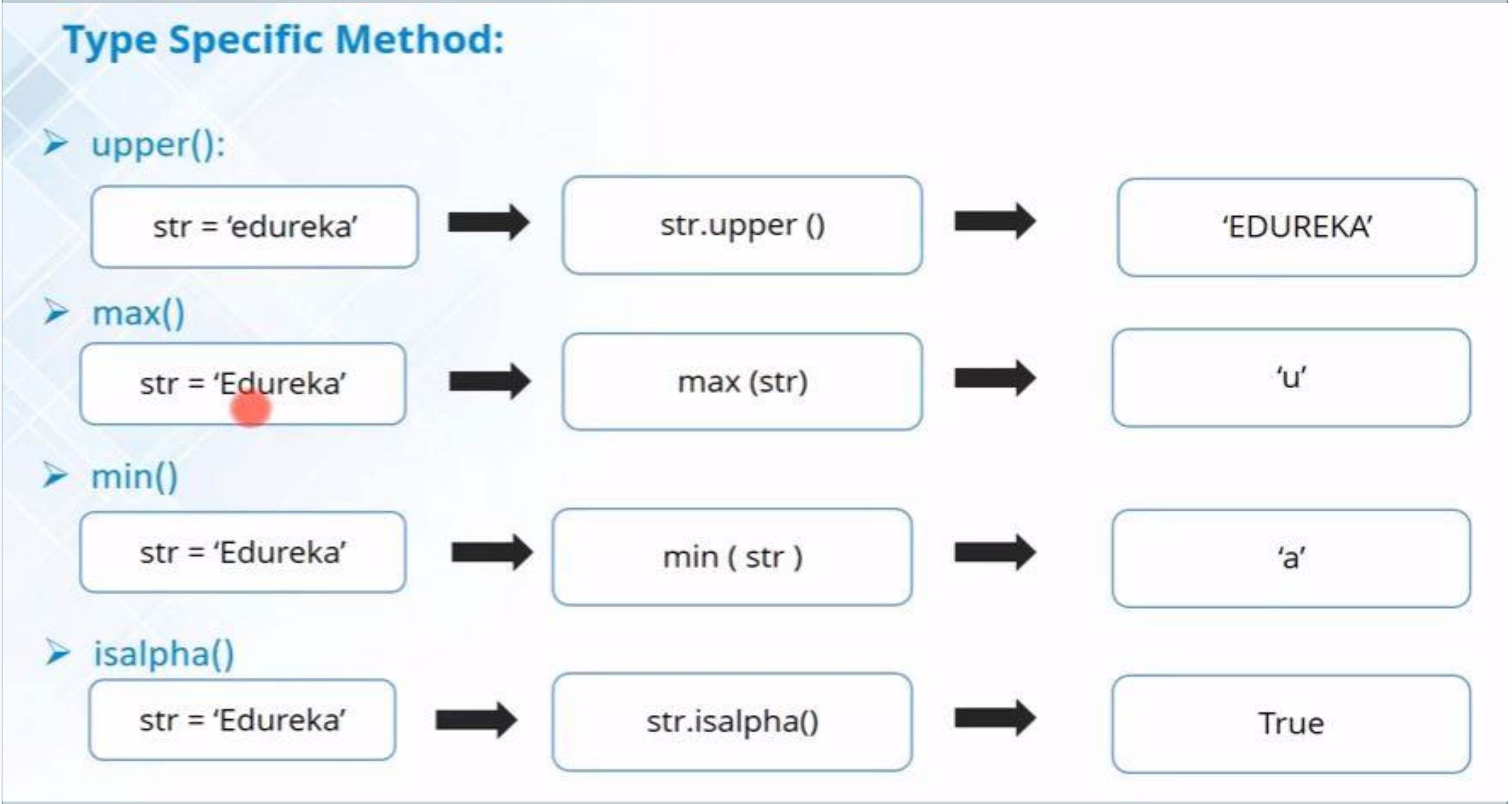
- split()

str = ‘E, d, u, r, e, k, a’ → s.split (‘,’) → [‘E’, ‘d’, ‘u’, ‘r’, ‘e’, ‘k’, ‘a’]

- count()

str = ‘edureka’ → str.count(‘e’, beg=0, end=6) → 2

Operation on String



Type Specific Method:

- upper():

```
str = 'edureka'  →  str.upper ()  →  'EDUREKA'
```

- max()

```
str = 'Edureka'  →  max (str)  →  'u'
```

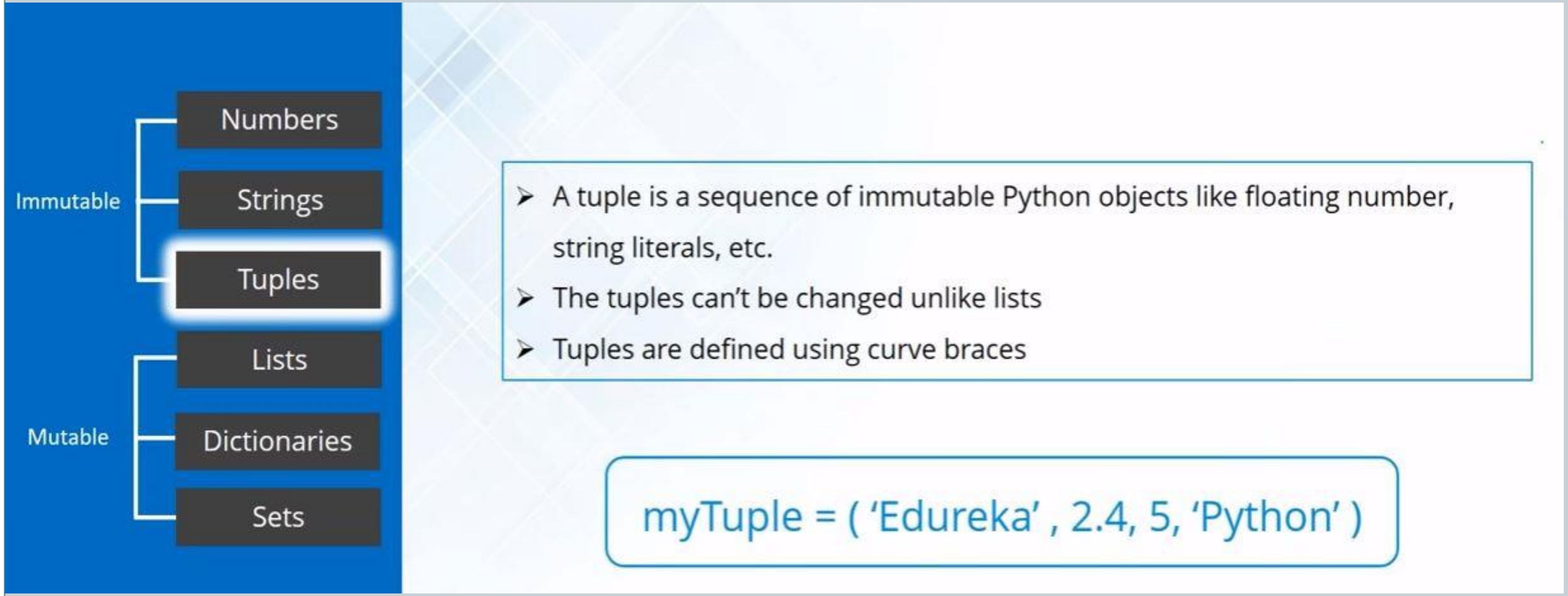
- min()

```
str = 'Edureka'  →  min ( str )  →  'a'
```

- isalpha()

```
str = 'Edureka'  →  str.isalpha()  →  True
```

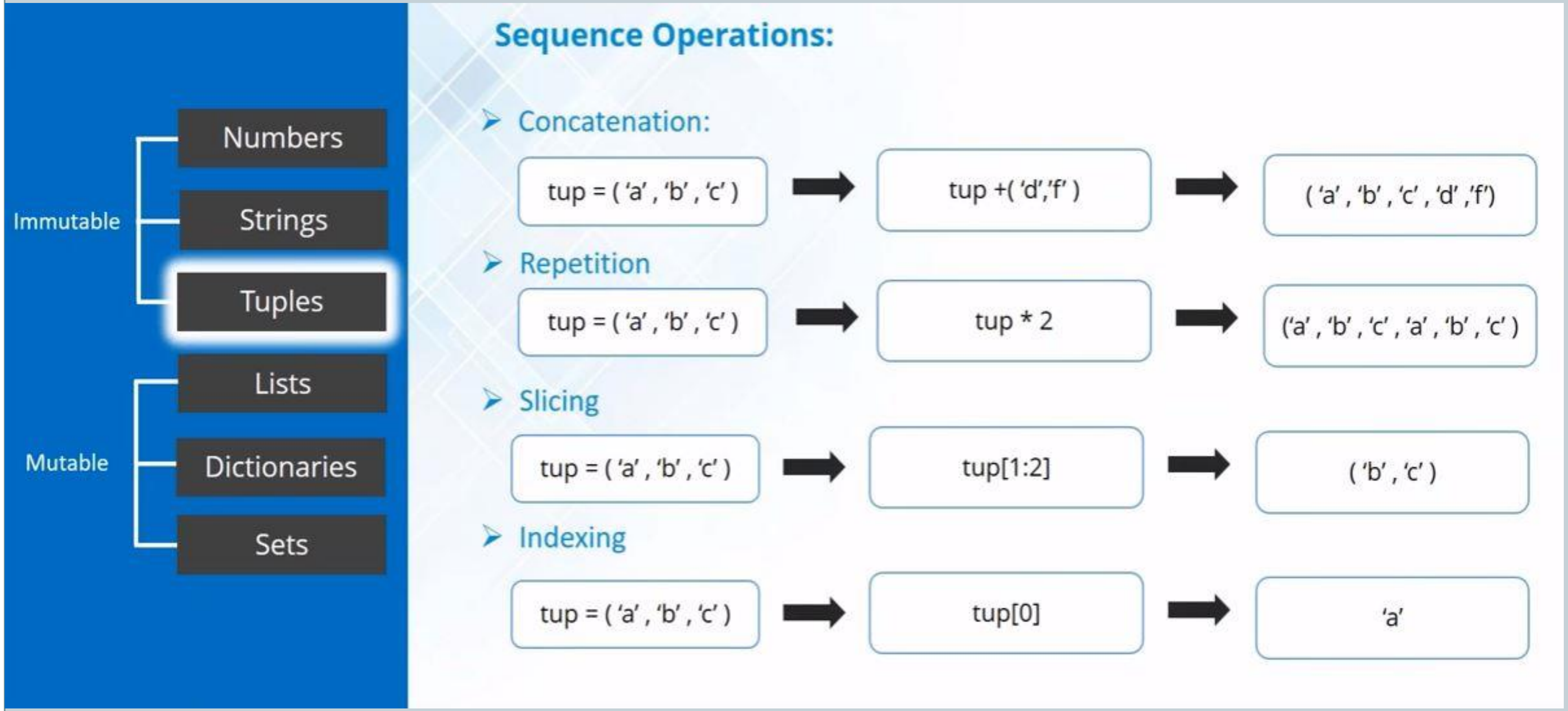
Tuple



- A tuple is a sequence of immutable Python objects like floating number, string literals, etc.
- The tuples can't be changed unlike lists
- Tuples are defined using curve braces

```
myTuple = ( 'Edureka' , 2.4, 5, 'Python' )
```

Operations on Tuple



Sequence Operations:

- Concatenation:

```
tup = ( 'a' , 'b' , 'c' )  →  tup +( 'd','f' )  →  ( 'a' , 'b' , 'c' , 'd' ,'f')
```

- Repetition

```
tup = ( 'a' , 'b' , 'c' )  →  tup * 2  →  ('a' , 'b' , 'c' , 'a' , 'b' , 'c' )
```

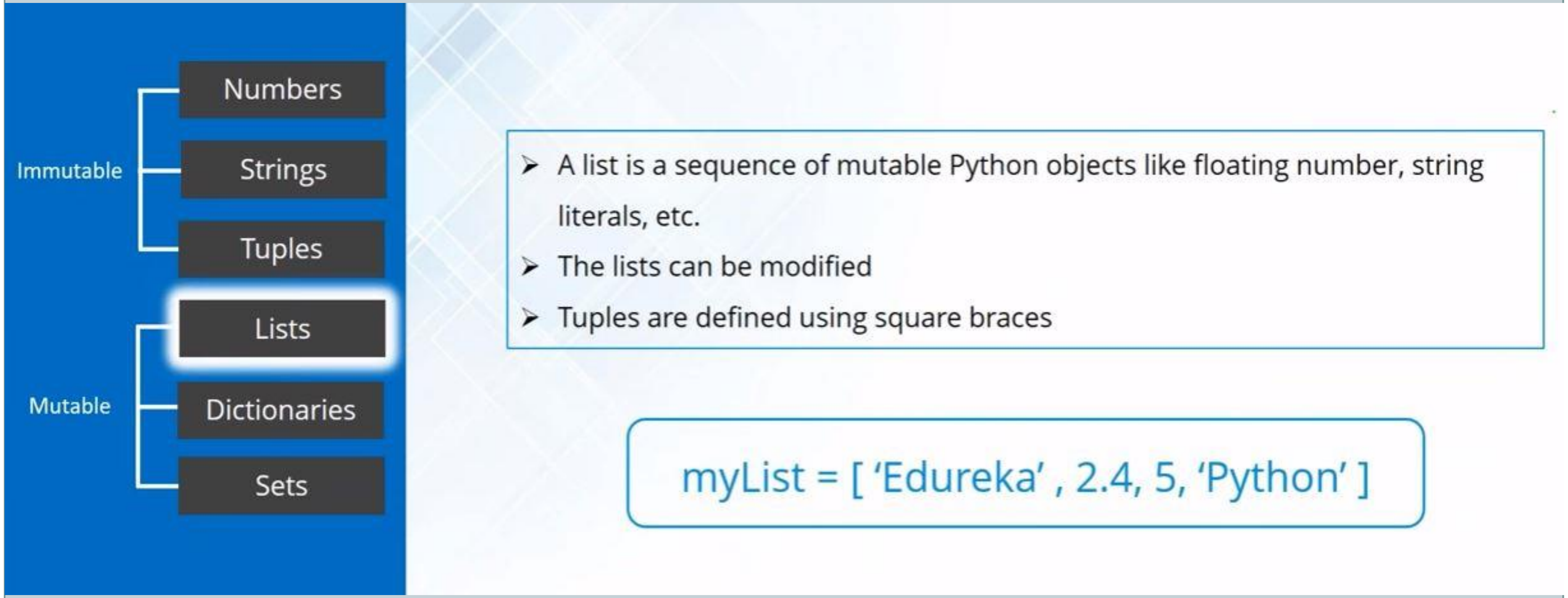
- Slicing

```
tup = ( 'a' , 'b' , 'c' )  →  tup[1:2]  →  ( 'b' , 'c' )
```

- Indexing

```
tup = ( 'a' , 'b' , 'c' )  →  tup[0]  →  'a'
```

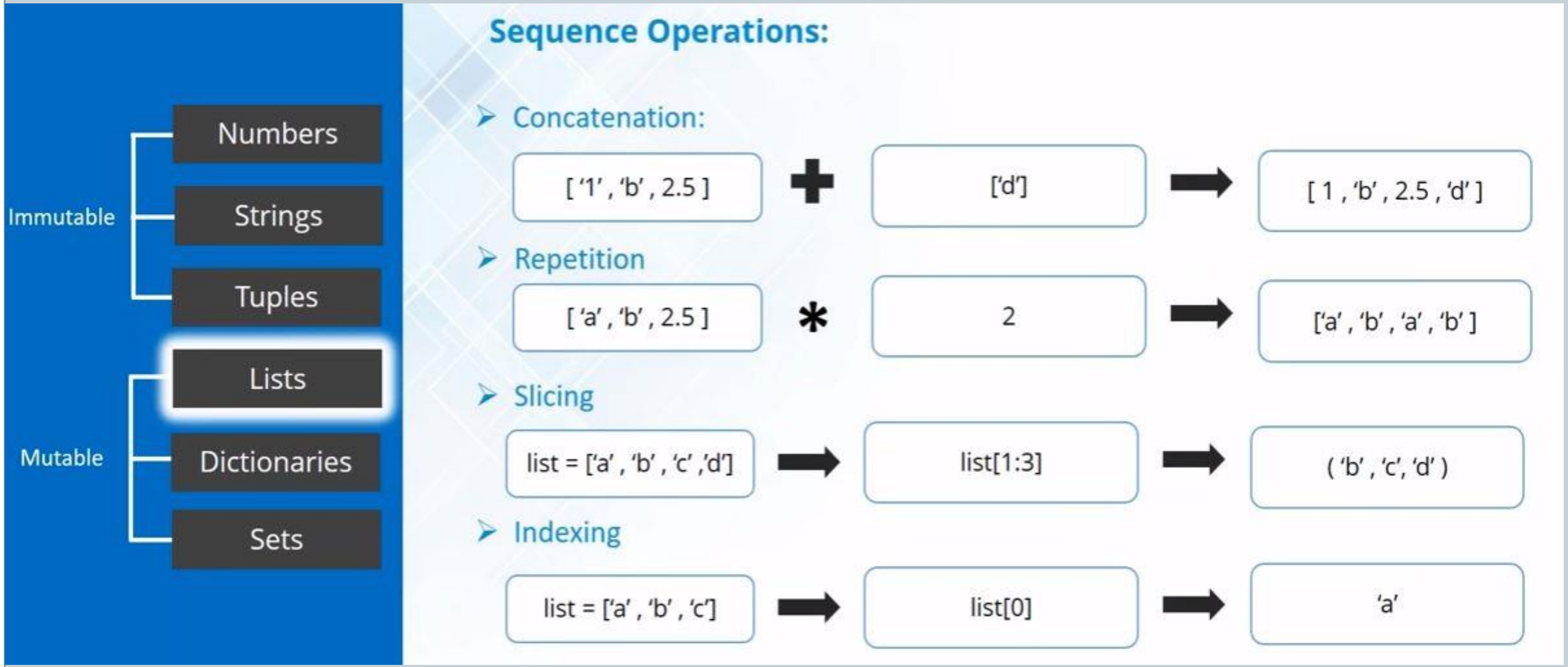
List



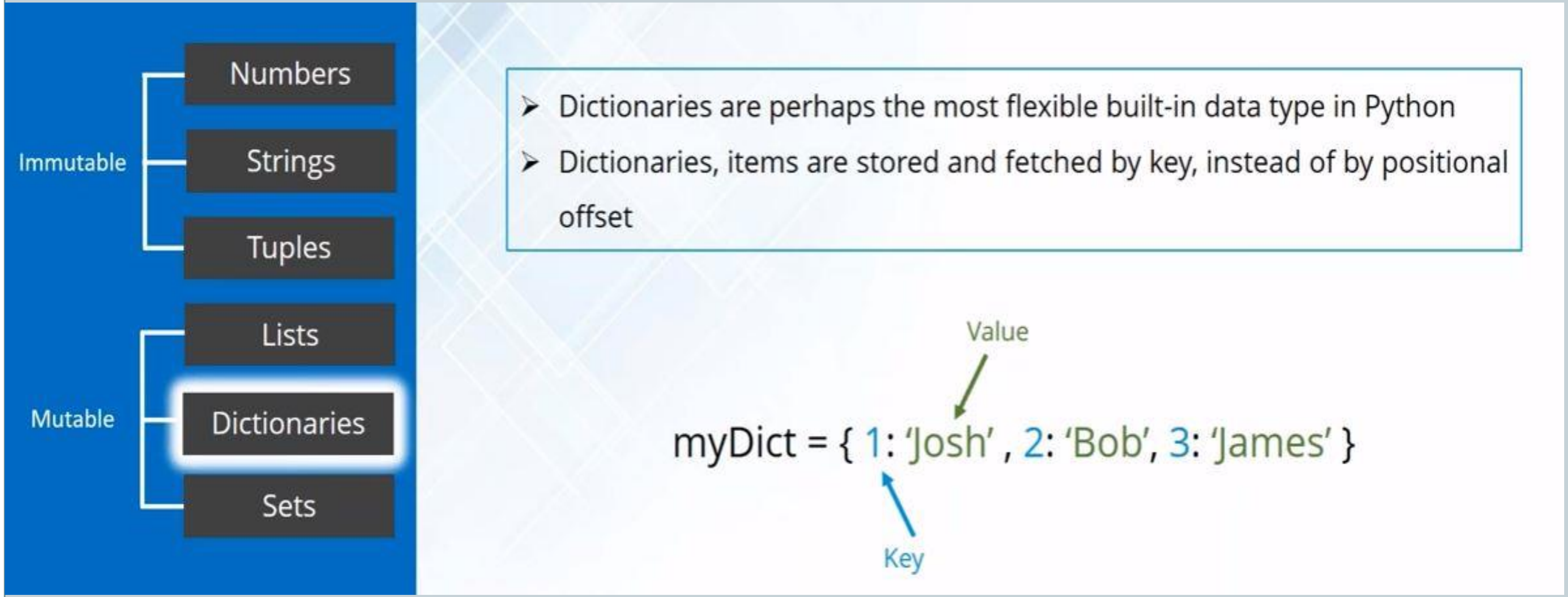
- A list is a sequence of mutable Python objects like floating number, string literals, etc.
- The lists can be modified
- Tuples are defined using square braces

```
myList = [ 'Edureka' , 2.4, 5, 'Python' ]
```

Operations on List



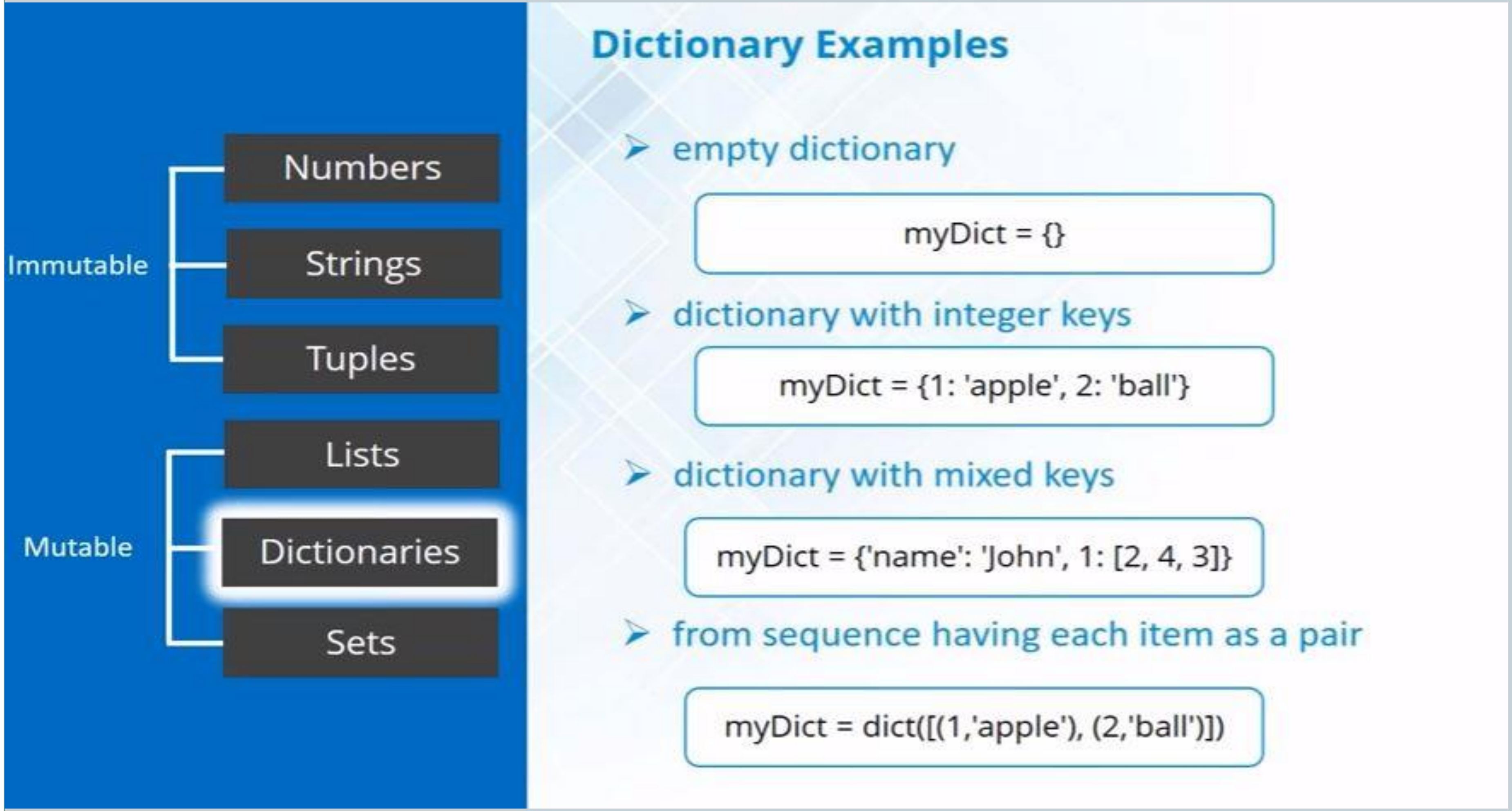
Dictionary



- Dictionaries are perhaps the most flexible built-in data type in Python
- Dictionaries, items are stored and fetched by key, instead of by positional offset

```
myDict = { 1: 'Josh' , 2: 'Bob', 3: 'James' }
```

Dictionary



Dictionary Examples

- empty dictionary

```
myDict = {}
```

- dictionary with integer keys

```
myDict = {1: 'apple', 2: 'ball'}
```

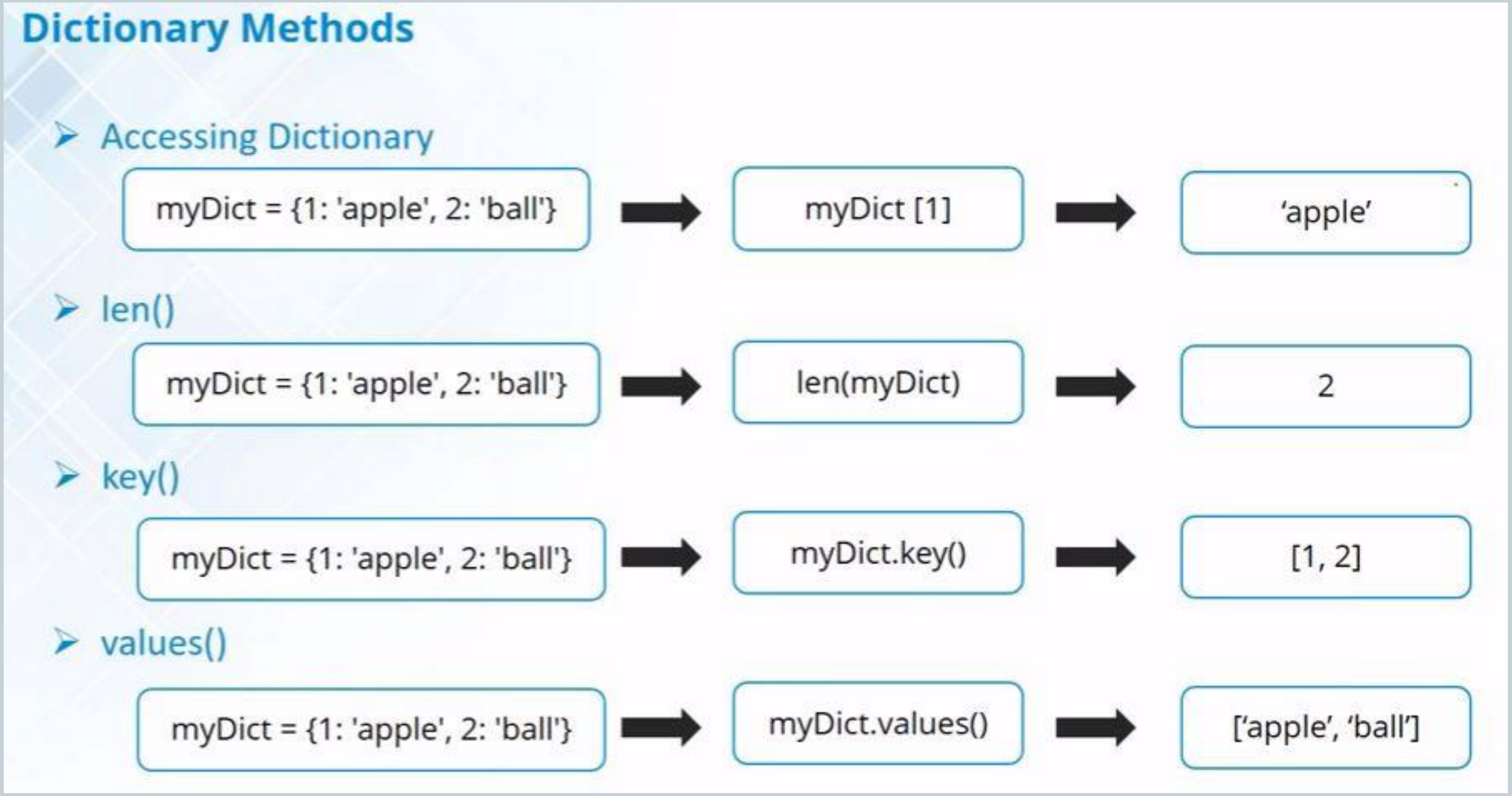
- dictionary with mixed keys

```
myDict = {'name': 'John', 1: [2, 4, 3]}
```

- from sequence having each item as a pair

```
myDict = dict([(1,'apple'), (2,'ball')])
```

Operations on Dictionary



Dictionary Methods

- Accessing Dictionary

```
myDict = {1: 'apple', 2: 'ball'}  →  myDict [1]  →  'apple'
```

- len()

```
myDict = {1: 'apple', 2: 'ball'}  →  len(myDict)  →  2
```

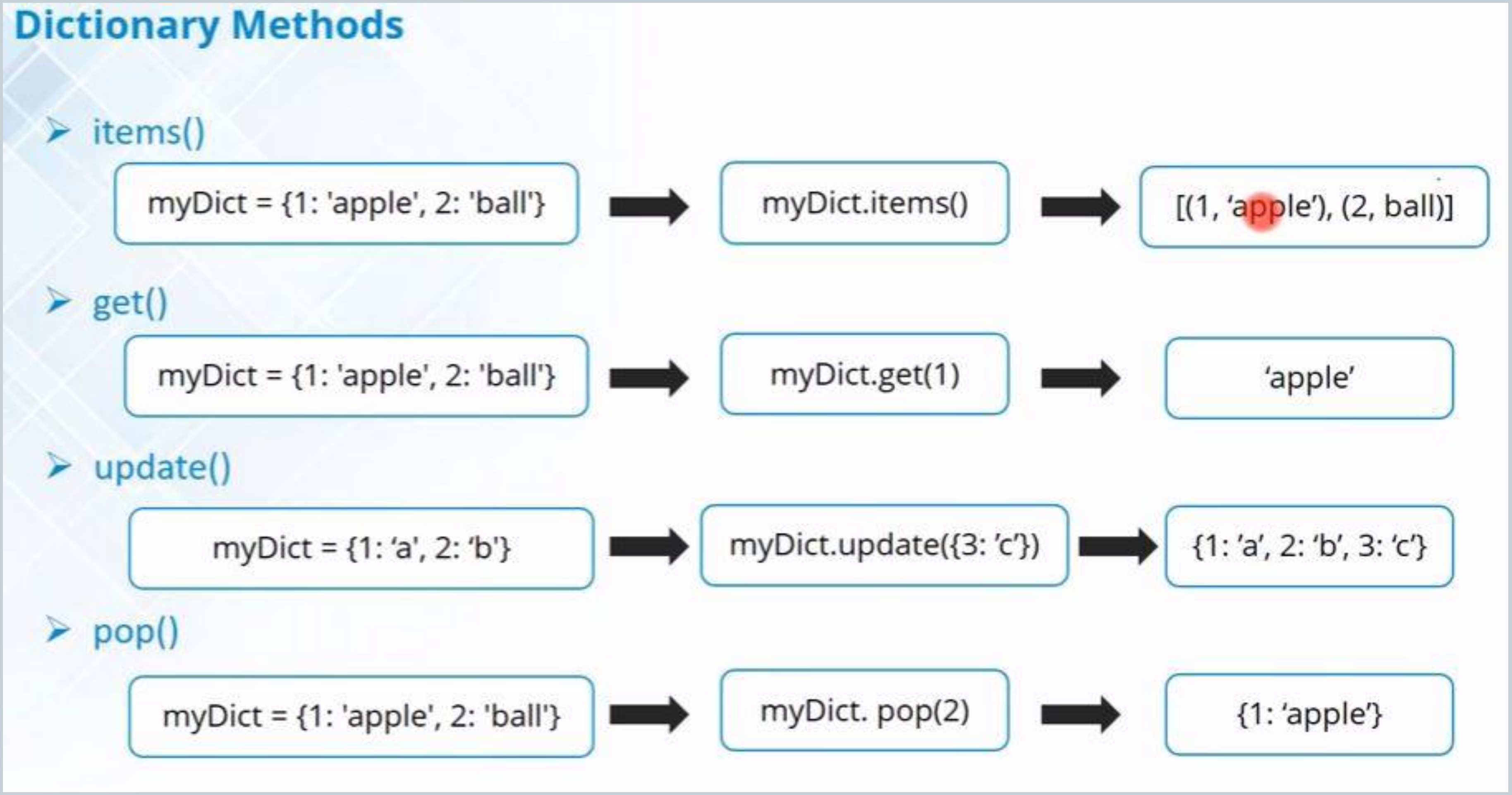
- key()

```
myDict = {1: 'apple', 2: 'ball'}  →  myDict.key()  →  [1, 2]
```

- values()

```
myDict = {1: 'apple', 2: 'ball'}  →  myDict.values()  →  ['apple', 'ball']
```

Operations on Dictionary



Dictionary Methods

- items()

  myDict = {1: 'apple', 2: 'ball'} → myDict.items() → [(1, 'apple'), (2, ball)]

- get()

  myDict = {1: 'apple', 2: 'ball'} → myDict.get(1) → 'apple'

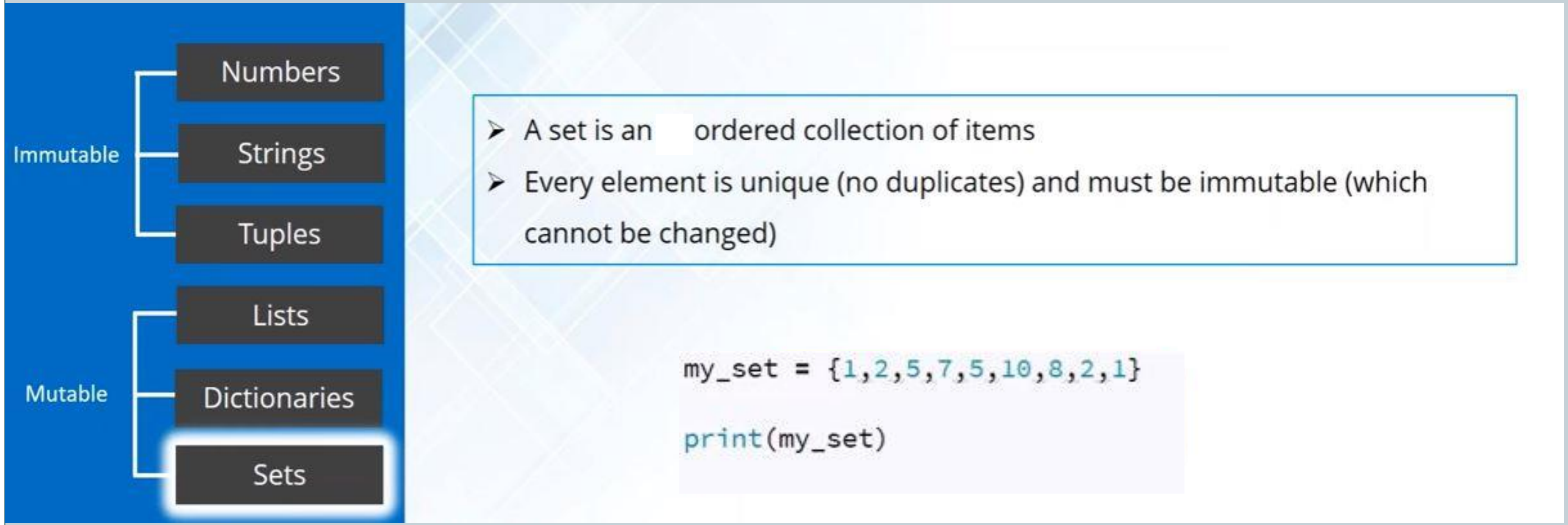
- update()

  myDict = {1: 'a', 2: 'b'} → myDict.update({3: 'c'}) → {1: 'a', 2: 'b', 3: 'c'}

- pop()

  myDict = {1: 'apple', 2: 'ball'} → myDict. pop(2) → {1: 'apple'}

Set

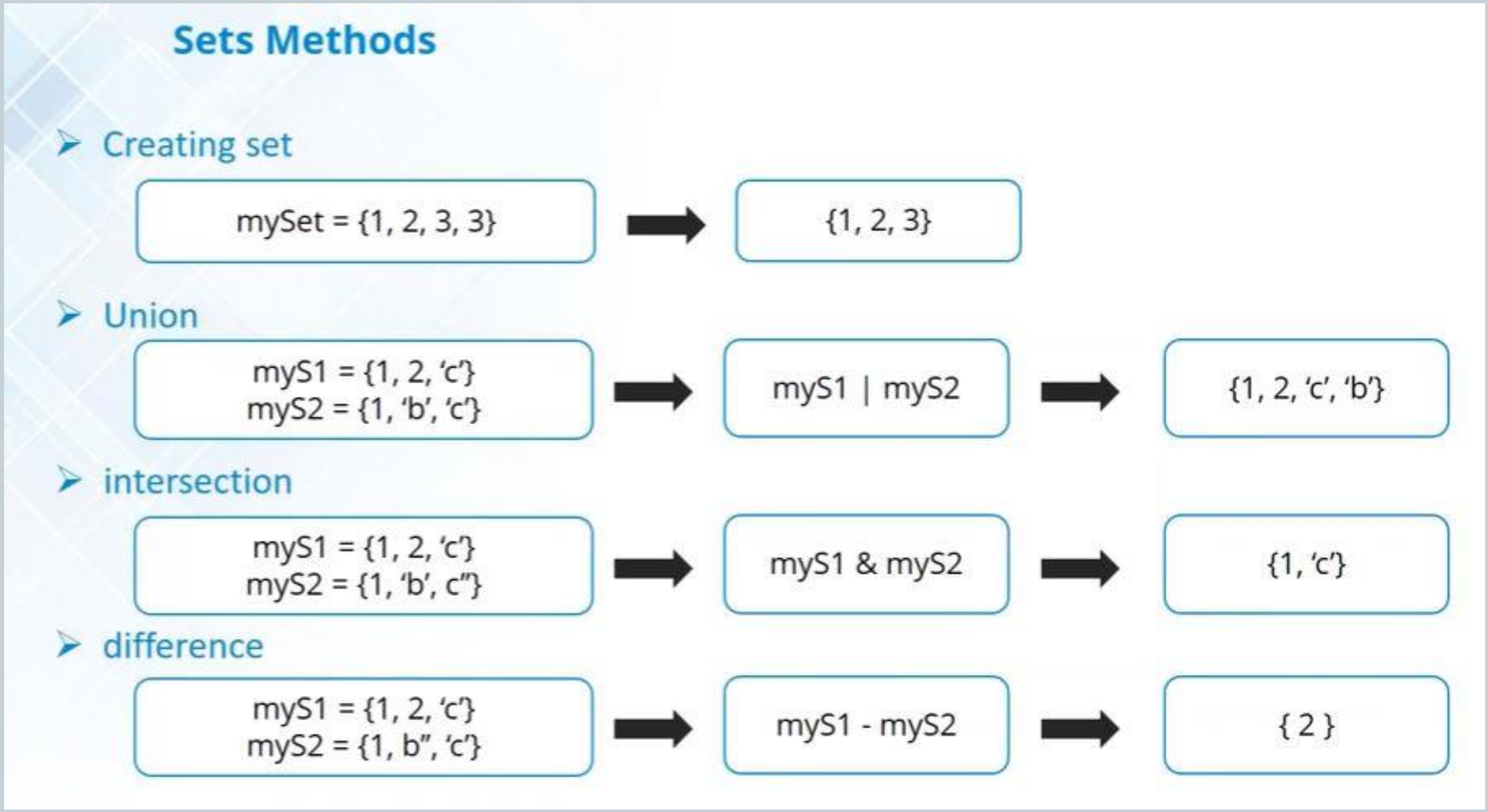


- A set is an ordered collection of items
- Every element is unique (no duplicates) and must be immutable (which cannot be changed)

```
my_set = {1,2,5,7,5,10,8,2,1}

print(my_set)
```

Operation on Set



Sets Methods

- Creating set

mySet = {1, 2, 3, 3} → {1, 2, 3}

- Union

mySet1 = {1, 2, 'c'}
mySet2 = {1, 'b', 'c'} → mySet1 | mySet2 → {1, 2, 'c', 'b'}

- intersection

mySet1 = {1, 2, 'c'}
mySet2 = {1, 'b', c"} → mySet1 & mySet2 → {1, 'c'}

- difference

mySet1 = {1, 2, 'c'}
mySet2 = {1, b", 'c'} → mySet1 - mySet2 → { 2 }