edureka!

Module 3 – Python Basics

COURSE OUTLINE
MODULE 03

1. Introduction to Data Science
2. Data Collection & Cleaning
3. Python Basics
4. Python Programming Concepts
5. Web Scraping
6. Scientific Computing using Arrays
7. Data Manipulation and Analysis
8. Visualizing Data

Topics

Following are the topics covered in this module:

- What is Python?
- Applications of Python
- Python Environment Setup
- Python Fundamentals
  - Syntax Rules
  - Indentation
  - Code Execution
  - Tokens
    - Keywords
    - Identifier
    - Literals
    - Operators
- Data Types
  - Boolean
  - Mutable and Immutable DataTypes
  - Mutable Data Types
    - Lists
    - Sets
    - Dictionary
  - Immutable Data Types
    - Numbers
    - Tuples
    - String

edureka!

Objectives

After completing this module, you should be able to:

- Understand importance of Python and its applications in Industry
- Use fundamentals of Python
- Understand Python framework and IDE
- Perform operations using Python concepts
- Classify Python Datatypes

edureka!

What Is Python?

edureka!

What Is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics

edureka!

Applications Of Python

edureka!

Applications Of Python

Companies Using Python

Google uses python as one of its three core languages, other two being Java and Golang

Dropbox uses Python across their platform for application development, infrastructure and operations

Music streaming giant **Spotify** uses Python for user data analysis and back-end services

Netflix uses Python extensively, using it for recommending movies, analysis of user statistics etc

The founders of **Quora** chose Python to implement their idea.

Reddit was recoded into Python from Lisp just after 6 months of its launch cause of its readability and vast libraries as per Steve Huffman and Alex Ohanian

edureka!

Fundamentals Of Python

edureka!

Fundamentals Of Python: Syntax Rules

01 Python is case sensitive. *Hello* is not equal to *hello*

02 Python does not have a **command terminator,** means no **;** or anything

03 Each line can have one statement at most. For multiple statements, you should use **semicolon ;**

04 **Comments** in Python start with **'#'** or **'''** . For single line comments, we use '#' and for multiline comments we use **'''** **(triple quotes)**

05 Python supports **multiline statements** i.e. **Line Continuation**. To use it, we put **backslash '\'** at the end of each line

```
1  x=1
2
3  y=2; z=3
4
5  #This is a comment
6  '''
7  This is a
8  multiline comment
9  with multiple lines
10 '''
11
12 123 + \
13 456 + \
14 789
```

edureka!

Fundamentals Of Python: Indentation

No braces to indicate blocks of code for class and function definitions or flow control

Blocks of code are denoted by line indentation, which is rigidly enforced

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount

Leading whitespace at the beginning of a logical line is used to compute the indentation level of the line, which in turn, is used to determine the grouping of statements

Fundamentals Of Python: Code Execution

Source

The structured code is written and saved with .py extension

Byte Code

The code is converted into Byte code for machine to understand

PVM

The Byte codes are executed

edureka!

Demo: Creating “Hello World” Program

edureka!

Creating “Hello World” Program

Every character in Python should be enclosed within single or double quotes

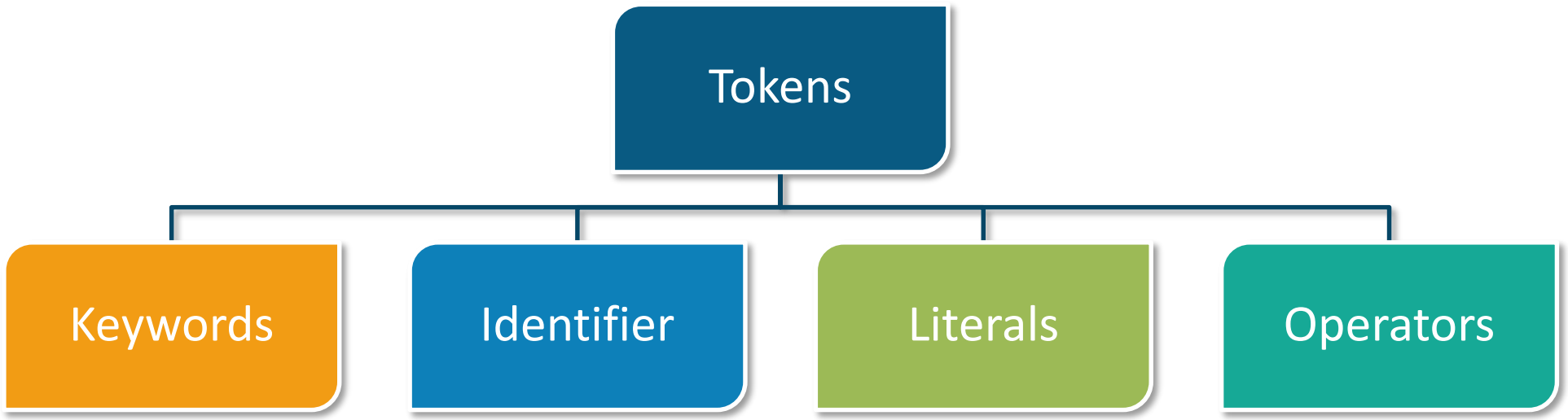
Output after running new.py

```
print('Hello World')
print("Welcome to Edureka")
```

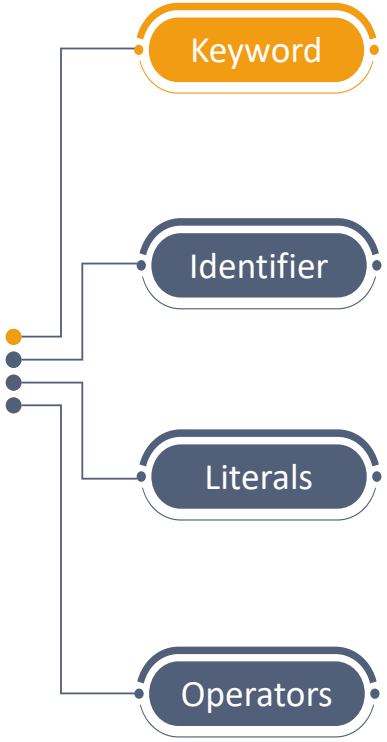
```
Hello World
Welcome to Edureka
```

Fundamentals Of Python: Tokens

Tokens are smallest lexical unit available in a program



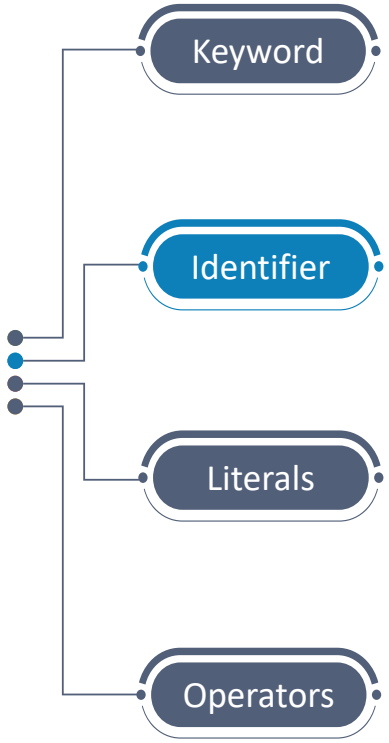
Tokens: Keyword



Keywords are reserved words that have special meaning or function in Python

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Tokens: Identifier



Identifiers are the names given to constants, variables, functions and user-defined classes

abc, q123, _2A, num_1

1abc, num%4, num 1

Rules for Identifier naming:

- The **first character** must be an *alphabet* or *underscore*(_)
- Except **first character** can be *alphanumeric (a-z, A-Z, 0-9 or _)*
- *Whitespaces* or *special characters* are not allowed (!, @, #, $, %, ^, &, *)
- Identifier name must not be a **keyword**
- Identifiers are case sensitive

edureka!

Identifiers – Naming Conventions

1. Class names start with an uppercase letter. All other identifiers start with a lowercase letter

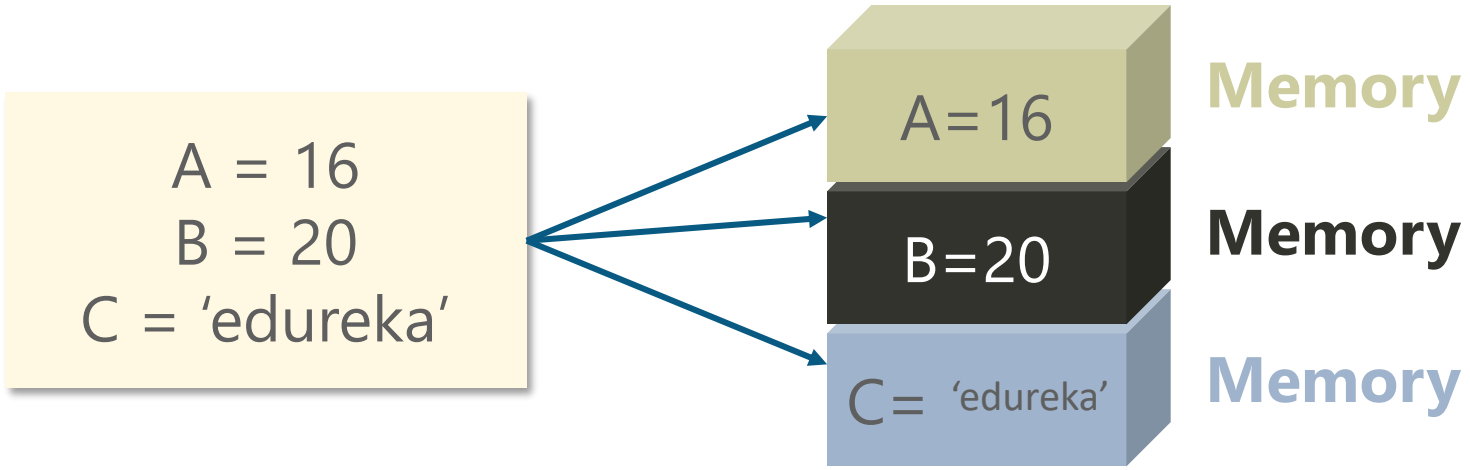
2. Starting an identifier with a single leading underscore indicates that the identifier is private

3. Starting an identifier with two leading underscores indicates a strongly private identifier

4. If the identifier also ends with two trailing underscores, the identifier is a language-defined special name

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory



edureka!

Variables (Contd.)

Consider the below screenshot, it explains how to assign a value to a *Variable*

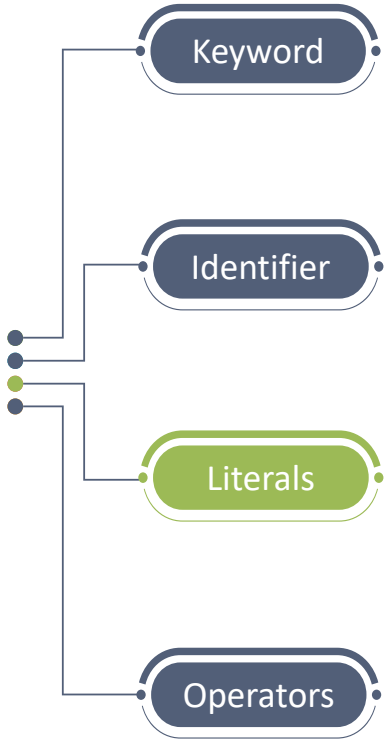
Assigning values 10 and edureka! to variables A and B respectively

```
A=10
B='edureka!'
print(A,B)
```

Output

```
10 edureka
```

Tokens: Literals



Literals are the data given to a variable. Python has various types of literals

Literals

- Integer: 122, 762, -3
- Float: 1.0, 2.341, 9.231
- Complex 1+2j, 1.0-6j, 88+22j

Numeric

- List: [],[1,34,343]
- Tuple: (),(78,34,12)

Collection

- ‘Hello’
- ‘abc123’

String

- [1,2,3,4]

Set

- {}, {‘a':1}

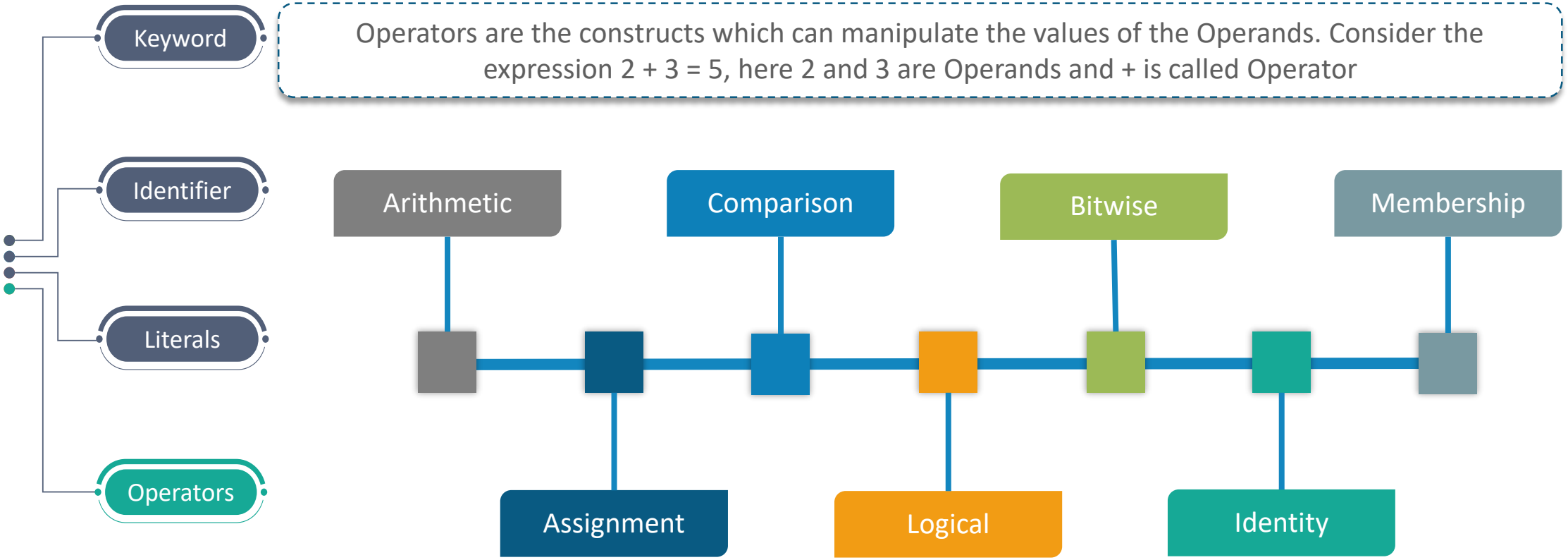
Dictionaries

- **True** or **False**

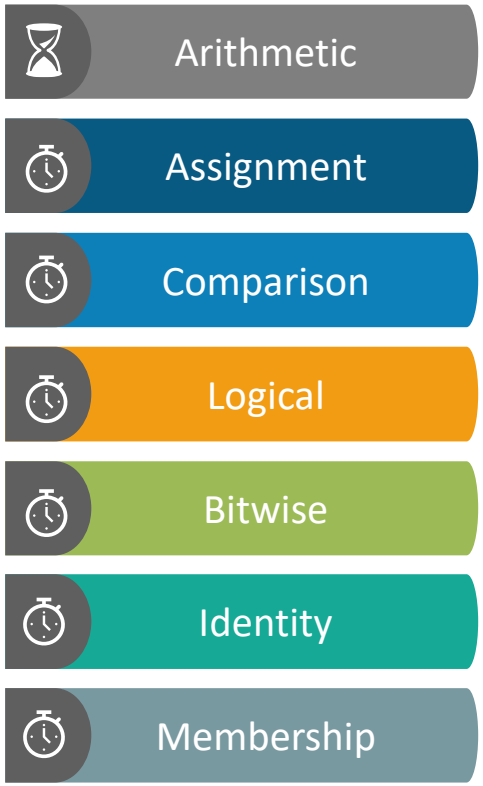
Boolean

edureka!

Tokens: Operators



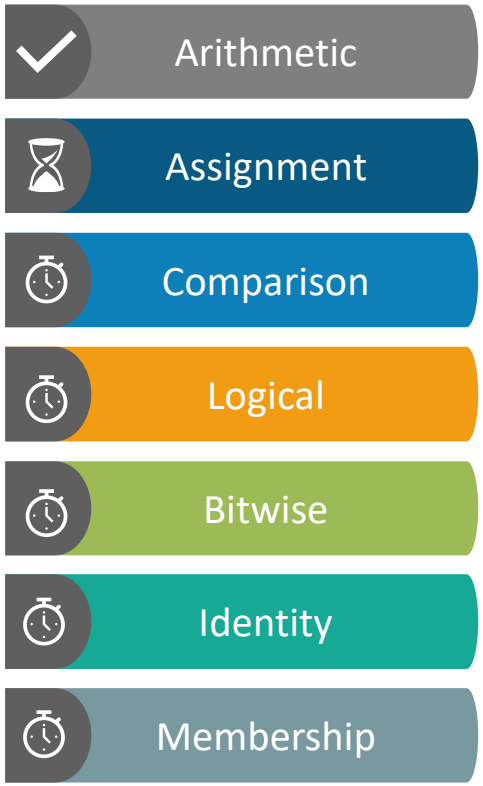
Tokens: Operators - Arithmetic



Addition	a + b
Subtraction	a – b
Multiplication	a * b
Division	a / b
Modulus	a % b
Exponent	a ** b
Floor Division	a // bx

edureka!

Tokens : Operators - Assignment



Assigns value from right to left	a = b
a = a + b	a + = b
a = a - b	a - = b
a = a*b	a * = b
a = a/b	a /= b
a = a**b	a** = b
a=a//b	a//=b

edureka!

Tokens : Operators - Comparision

Equal To	a == b
Not Equal To	a != b
Greater Than	a > b
Less Than	a < b
Greater Than Equal To	a >= b
Less Than Equal To	a <= b

edureka!

Tokens : Operators - Logical

- Arithmetic
- Assignment
- Comparison
- Logical
- Bitwise
- Identity
- Membership

a and b → Returns **True** if both a and b are **True**

a or b → Returns **True** if either or both are **True**

not a → Returns **True** if a is **False**, **True** otherwise

edureka!

Tokens : Operators - Bitwise

Binary AND	a & b
Binary OR	a | b
Binary XOR	a ^ b
Binary NOT	a ~ b
Binary Left Shift	a <<
Binary Right Shift	a >> b

edureka!

Tokens : Operators - Identity

is	Evaluates to TRUE if the variables on either side of the operator point to the same object and FALSE otherwise
is not	Evaluates to FALSE if the variables on either side of the operator point to the same object and TRUE otherwise

Tokens : Operators - Membership

in	Evaluates to TRUE if it finds a variable in the specified sequence and FALSE otherwise
not in	Evaluates to TRUE if it does not find a variable in the specified sequence and FALSE otherwise

edureka!

Python Data Types

edureka!

Python Data Type: Boolean

- The bool datatype represents truth values from logic.
- In Python, to represent a *truth value* we use literal **True** and for a *false value* we use **False**.

```
1  x=2
2  y=1
3  print('hello'=='Hello')
4  print((x+y)==3)
5  print(x>y)
6  x=y
7  print(x!=y)
```

```
False
True
True
False
```

Output

```
1  x=1.1+2.2
2  y=3.3
3  x==y
```

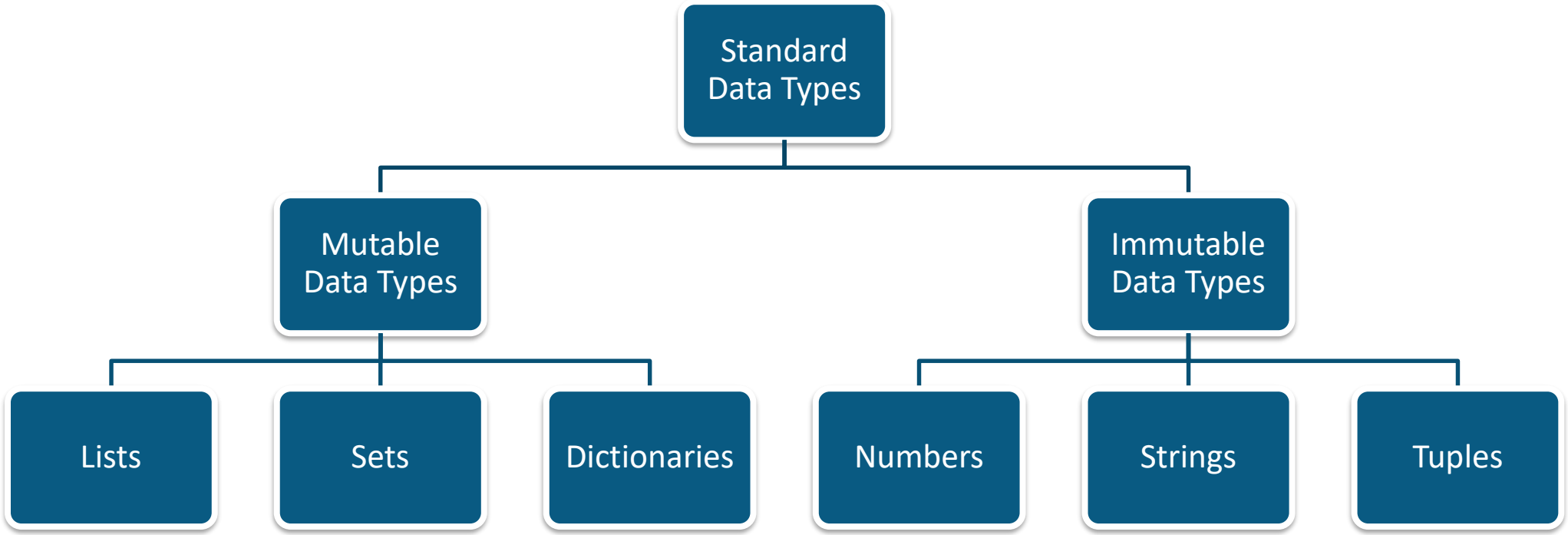
```
False
```

Mutable And Immutable Datatypes

edureka!

Mutable And Immutable DataTypes

- **Mutable Sequences** are sequences that *can be changed.*
- **Immutable Sequences** are sequences that *cannot be changed.*



edureka!

Mutable Data Types

edureka!

Mutable Sequences: Lists

List is an ordered set of elements enclosed within square brackets. The main differences between Lists and Tuples are:

- Lists are enclosed in brackets[] and Tuples are enclosed within parenthesis()
- Lists are Mutable and Tuples are Immutable
- Tuples are faster than Lists

```
1  # empty list
2  l1 = []
3  # list of integers
4  l2 = [1, 2, 3]
5  # list with mixed datatypes
6  l3 = [1, "edureka", 3.14]
7  print(l1,l2,l3)
```

Lists are enclosed within square brackets

```
[] [1, 2, 3] [1, 'edureka', 3.14]
```

Output

Lists: Indexing

- Lists can be accessed using **index operator []**
- Index starts from **0.** So, a list having **10** elements will have index from **0** to **9**
- Index must be an Integer value, otherwise interpreter will raise **TypeError**

```
1  my_list = ['e','d','u','r','e','k','a']
2  print(my_list[0])
3  print(my_list[2])
4  print(my_list[-2])
5  # Nested List
6  n_list = ["Hello!", ["Welcome",'To',2019]]
7  # Nested indexing
8  print(n_list[0])
9  print(n_list[1][2])
10 print(n_list[-1])
```

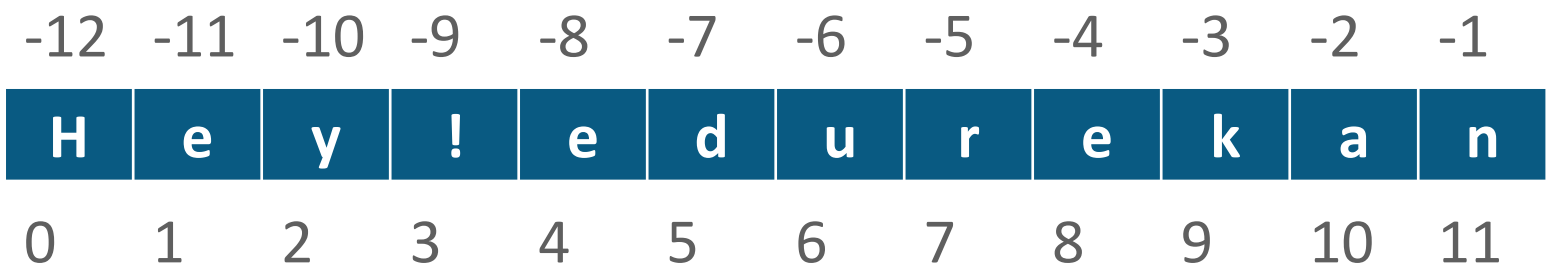
Python also supports negative indexing

```
e
u
k
Hello!
2019
['Welcome', 'To', 2019]
```

Output

Lists: Slicing

To access a range of items in a list we can using the slicing operator(colon :)




```
my_list=['H','e','y','!','e','d','u','r','e','k','a','n']
# elements beginning to 4th
print(my_list[:-8])
# elements 6th to end
print(my_list[4:])
# elements 3rd to 5th
print(my_list[2:5])
# elements beginning to end
print(my_list[:])
```

```
['H', 'e', 'y', '!']
['e', 'd', 'u', 'r', 'k', 'a', 'n']
['y', '!', 'e']
['H', 'e', 'y', '!', 'e', 'd', 'u', 'r', 'k', 'a', 'n']
```

Output

edureka!

Lists: Insertion

append()	Add an element to the end of the list
extend()	Add all elements of a list to another list
insert()	Insert an item at the defined index

```
1  my_list=[1,2,3,4]
2  print(my_list)
3  my_list.append(5)
4  print(my_list)
5  my_list.extend([7,8,9])
6  print(my_list)
7  my_list.insert(5,6)
8  print(my_list)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Output

Lists: Deletion

remove()	Removes an item from the list
pop()	Removes and returns an element at the given index
clear()	Removes all items from the list

```
1  my_list=[1,2,3,4,5,6]
2  my_list.remove(3)
3  print(my_list)
4  print(my_list.pop(4))
5  my_list.clear()
6  print(my_list)
7  my_list=[1,2,3,4,5,6]
8  del my_list[2]
9  print(my_list)
10 del my_list
11 print(my_list)
```

del can be used for any Data Type

Will raise error as we have deleted whole list

```
[1, 2, 4, 5, 6]
6
[]
[1, 2, 4, 5, 6]

---------------------------------------
NameError
<ipython-input-78-707ae3b1f483> in <modul
      9 print(my_list)
     10 del my_list
---> 11 print(my_list)

NameError: name 'my_list' is not defined
```

Output

edureka!

Lists: Methods

index()	Returns the index of the first matched item
count()	Returns the count of number of items passed as an argument
sort()	Sort items in a list in ascending order
reverse()	Reverse the order of items in the list
copy()	Returns a shallow copy of the list

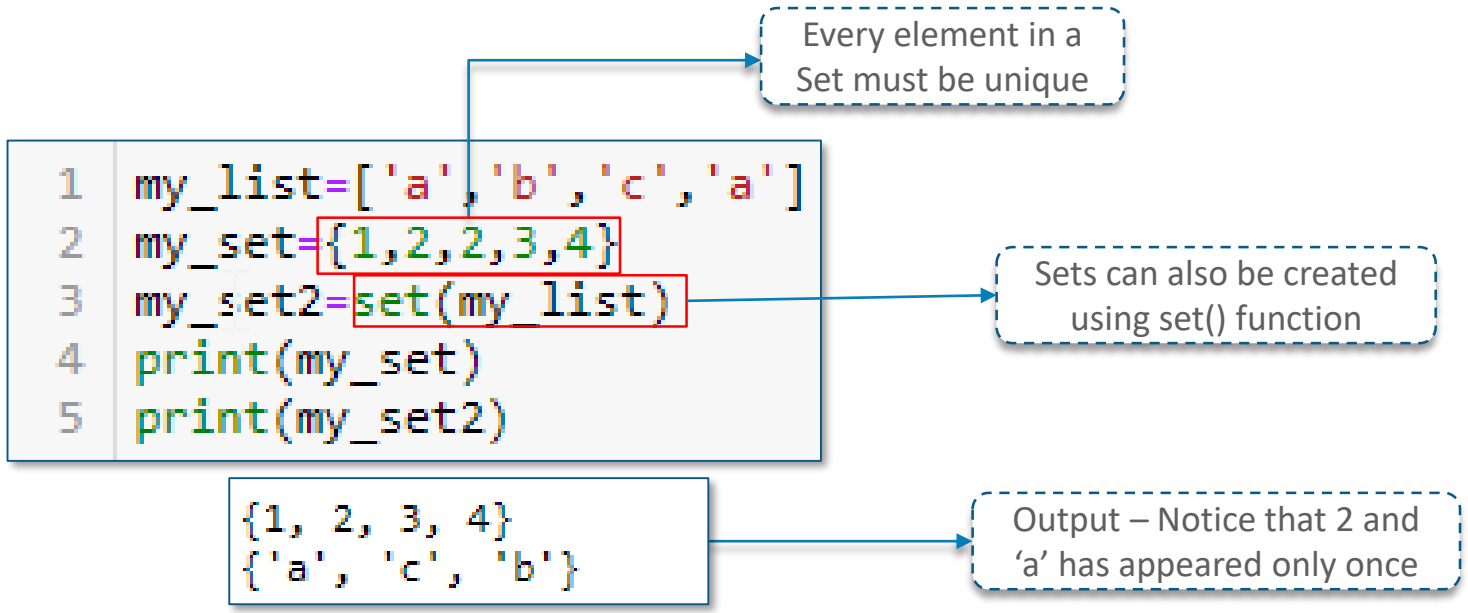
```
1  my_list=[1,4,3,4,5,2]
2  print(my_list.index(4))
3  print(my_list.count(4))
4  my_list.sort()
5  print(my_list)
6  my_list.reverse()
7  print(my_list.copy())
```

```
1
2
[1, 2, 3, 4, 4, 5]
[5, 4, 4, 3, 2, 1]
```

Output

Mutable Data Types: Sets

- A set is an unordered collection of items
- Every element is **unique** in a set
- A set is created by placing all the items (elements) inside **curly braces {}**, separated by comma(**,**)
- Sets do not support indexing



Sets: Insertion And Deletion

- We can insert elements in a set using **add()** method
- To insert set, tuple or a list in a set, we use **update()** method
- To remove elements, we use **discard()** and **remove()** method
- **pop()** method removes the item as well as return that element
- We use **clear()** method to remove all items from a set

```
1  my_set = {1,5}
2  print(my_set)
3
4  my_set.add(3)
5  print(my_set)
6
7  my_set.update([2,4])
8  print(my_set)
9
10 my_set.update([5,6], {2,4,8})
11 print(my_set)
```

```
{1, 5}
{1, 3, 5}
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6, 8}
```

Output

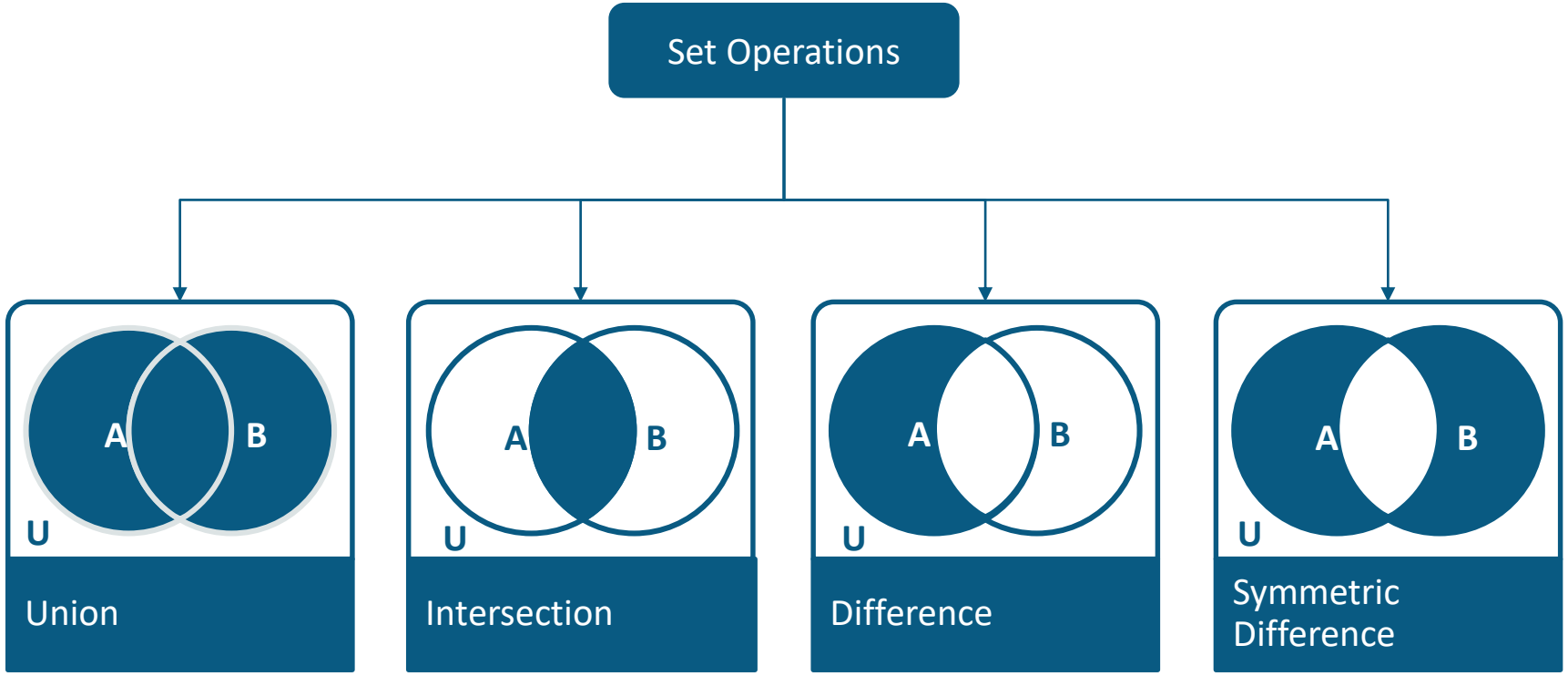
```
1  my_set={1, 2, 3, 4, 5, 6, 8}
2  print(my_set)
3
4  my_set.discard(1)
5  my_set.remove(2)
6  print(my_set)
7  print(my_set.pop())
8  my_set.clear()
9  print(my_set)
```

```
{1, 2, 3, 4, 5, 6, 8}
{3, 4, 5, 6, 8}
3
set()
```

Output

edureka!

Sets: Operations



Sets: Operations(Contd.)

Python provides **built-in functions** as well as **operators** for set operations

Set Operation	Operator	Function
Union	|	union()
Intersection	&	intersection()
Difference	-	difference()
Symmetric Difference	^	symmetric_difference()

```
1  set_a={1,2,4,5,6}
2  set_b={3,5,6,7,8}
3  print(set_a|set_b)
4  print(set_a & set_b)
5  print(set_a.difference(set_b))
6  print(set_a.symmetric_difference(set_b))
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{5, 6}
{1, 2, 4}
{1, 2, 3, 4, 7, 8}
```

Output

Mutable Data Types: Dictionary

- Python dictionary is an unordered collection of items
- Dictionaries contain **key-value** pairs
- Each key is separated from its value by a **colon (:)**, the items are separated by comma
- Dictionaries are enclosed by **curly braces**(**{}**)

Dictionaries can also be defined using **dict()** method

```
1  my_dict = {}
2  # dictionary with integer keys
3  my_dict = {1: 'one', 2: 'two'}
4  print(my_dict)
5  # dictionary with mixed keys
6  my_dict = {'Name': 'Sameer', 1: [2,3,4]}
7  print(my_dict)
8  my_dict = dict([('Class',10), ('Age',14)])
9  print(my_dict)
10 print(my_dict['Class'])
11 print(my_dict.get('Age'))
```

Elements can be accessed using **get()** or normal indexing

```
{1: 'one', 2: 'two'}
{'Name': 'Sameer', 1: [2, 3, 4]}
{1: 'one', 2: 'two'}
{'Class': 10, 'Age': 14}
```

Output

edureka!

Dictionary: Insertion And Deletion

- To update a dictionary we simply use assignment operator
- If key is present, then value is updated else new key-value is generated
- **pop()** is removes element by returning corresponding value
- Whereas **popitem()** removes arbitrary item and return it

```
1  my_dict = {'Name':'Ravi', 'age': 26}
2  my_dict['age'] = 27
3  print(my_dict)
4  # add item
5  my_dict['address'] = 'Bengalore'
6  print(my_dict)
7
8  #Deletion
9  print(my_dict.pop('address'))
10 print(my_dict.popitem())
11 print(my_dict)
```

```
{'Name': 'Ravi', 'age': 27}
{'Name': 'Ravi', 'age': 27, 'address': 'Bengalore'}
Bengalore
('age', 27)
{'Name': 'Ravi'}
```

Output

Dictionary: Methods

fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).
items()	Return a new view of the dictionary's items (key, value)
keys()	Return a new view of the dictionary's keys
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys
values()	Return a new view of the dictionary's values

```
1  keys=['Name','ID','Class']
2  my_dict=dict.fromkeys(keys)
3  print(my_dict)
4  my_dict['Name']='Sanjay'
5  my_dict['ID']=1234
6  my_dict['Class']='A'
7  print(my_dict.items())
8  print(my_dict.keys())
9  new_dict={'Name':'Aru','ID':456,'Class':'S'}
10 my_dict.update(new_dict)
11 print(my_dict.values())
```

```
{'Name': None, 'ID': None, 'Class': None}
dict_items([('Name', 'Sanjay'), ('ID', 1234), ('Class', 'A')])
dict_keys(['Name', 'ID', 'Class'])
dict_values(['Aru', 456, 'S'])
```

Output

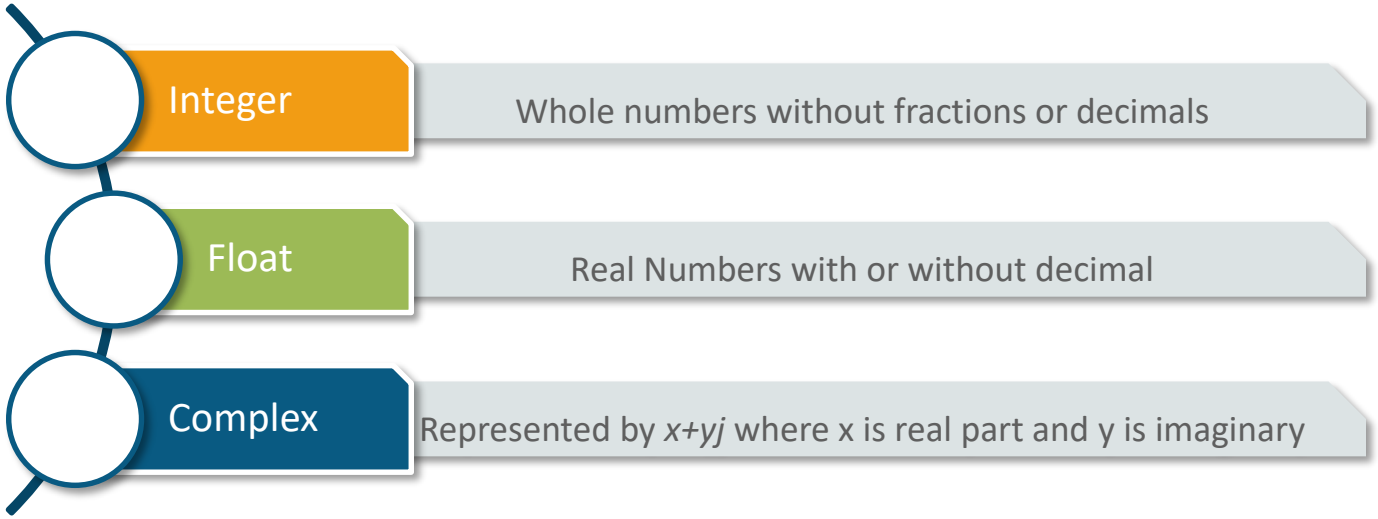
edureka!

Immutable Datatypes

edureka!

Immutable Data Types: Numbers In Python

Python supports integer, float and complex numbers defined as a class.




```
1  x=10 #Integer
2  y=10.1 #Float
3  z=1+2j #complex
4
```

Numbers In Python: Number System

Integers can be represented in different number systems using prefixes and functions for variables

Number System	Prefix	Function
Binary	0**b** or 0**B**	bin()
Octal	0**o** or 0**O**	oct()
Hexadecimal	0**x** or 0**X**	hex()

```
1  x=10
2
3  #Binary Base-2
4  print(0b11)
5
6  #Octal Base-8
7  print(0o11)
8
9  #Hexadecimal Base-16
10 print(0x11)
11
12 #variable
13 print('Variable')
14
15 #Binary Base-2
16 print(bin(x))
17
18 #Octal Base-8
19 print(oct(x))
20
21 #Hexadecimal Base-16
22 print(hex(x))
23
```

```
3
9
17
Variable
0b1010
0o12
0xa
```

Output

Numbers In Python: Type Conversion

Type conversion can be done using respective functions.

```
1  #Convert to Integer
2  print(int(123.54))
3  print(int('14'))
4
5  #Convert to float
6  print(float(7))
7  print(float('-4.5'))
8
9  #Convert to complex
10 print(complex(11))
11 print(complex('9-5j'))
```

```
123
14
7.0
-4.5
(11+0j)
(9-5j)
```

Output

Numbers In Python: Type and Instance

We can check the type of variable or the value or the function to which class it belongs using **type()** and **isintance()** function

```
1  x=10
2  y=22.33
3  z=44+55j
4
5
6  #Type
7  print(type(x))
8  print(type(y))
9  print(type(z))
10
11 #Check instance
12 print(isinstance(x,int))
13 print(isinstance(x,float))
14
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
True
False
```

Output

Immutable Data Types: Strings

- A string is a sequence of characters
- Python allows for either pairs of single or double or triple quotes
- Python does not support a character type; these are treated as strings of length one

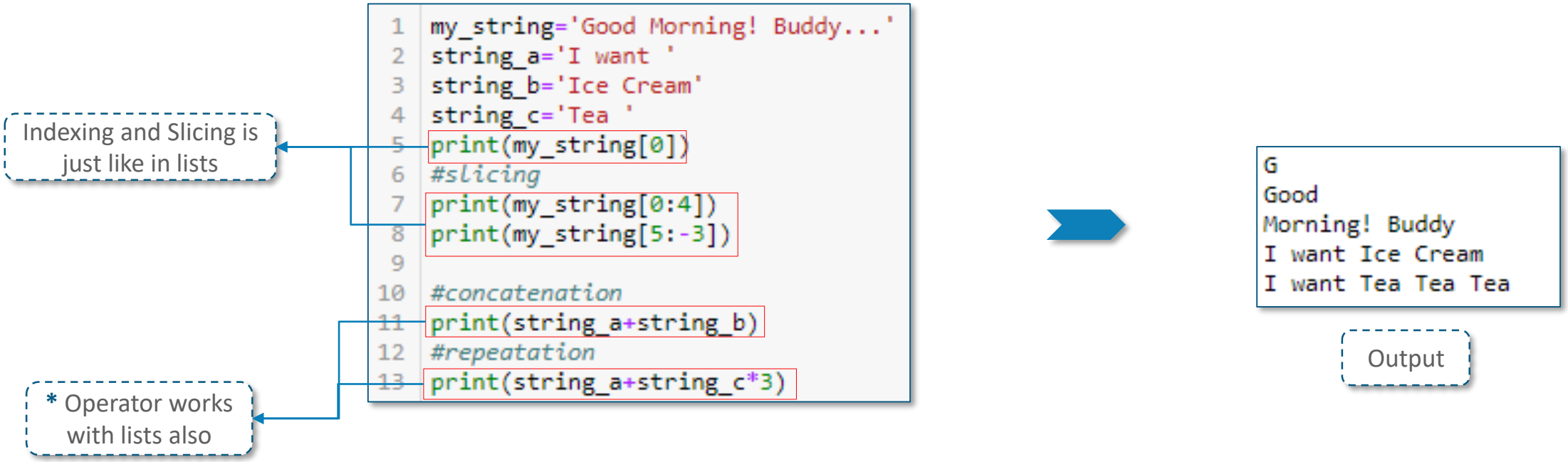
```
my_string = 'Hello'
print(my_string)

my_string = "Hey!"
print(my_string)

my_string = '''edurekan'''
print(my_string)

# triple quotes string can extend to multiple lines
my_string = """Hello, welcome to
           edureka"""
print(my_string)
```

```
Hello
Hey!
edurekan
Hello, welcome to
           edureka
```

Output

Strings: Indexing And Operations

- Indexing and slicing is similar to lists in Strings
- Operator (**+**) is used for concatenation of two strings
- Operator (*****) is used to print an element multiple times



edureka!

Immutable Data Types: Tuples

- A tuple consists of various items and they may be of different types
- Items are separated by comma(,) and enclosed within parentheses()
- Tuples are similar to lists except they are immutable

```
1  my_tuple = ()
2  print(my_tuple)
3
4  my_tuple = (1, 2, 3)
5  print(my_tuple)
6
7  # tuple with mixed datatypes
8  my_tuple = (1, "Hello!", 3.4)
9  print(my_tuple)
10
11 # nested tuple
12 my_tuple = ("Kitty", [1,2,3], ('a', 'b', 3))
13 my_tuple[1][2]=5
14 print(my_tuple)
```

If a tuple holds mutable object, they can be changed

```
()
(1, 2, 3)
(1, 'Hello!', 3.4)
('Kitty', [1, 2, 5], ('a', 'b', 3))
```

Output

Summary

What Is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics

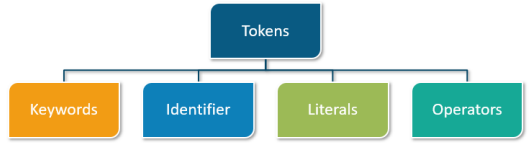
edureka!

Applications Of Python

edureka!

Fundamentals Of Python: Tokens

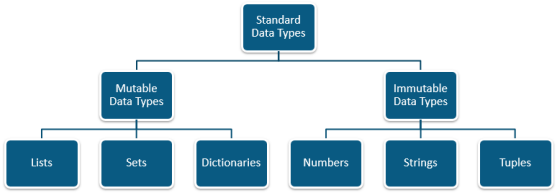
Tokens are smallest lexical unit available in a program



edureka!

Mutable And Immutable DataTypes

- **Mutable Sequences** are sequences that *can be changed.*
- **Immutable Sequences** are sequences that *cannot be changed.*



edureka!

Mutable Sequences: Lists

List is an ordered set of elements enclosed within square brackets. The main differences between Lists and Tuples are:

- Lists are enclosed in brackets[] and Tuples are enclosed within parenthesis()
- Lists are Mutable and Tuples are Immutable
- Tuples are faster than Lists

```
# empty list
l1 = []
# list of integers
l2 = [1, 2, 3]
# list with mixed datatypes
l3 = [1, "edureka", 3.14]
print(l1,l2,l3)
```

Lists are enclosed within square brackets

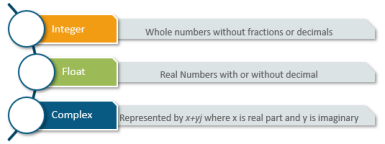
```
[] [1, 2, 3] [1, 'edureka', 3.14]
```

Output

edureka!

Immutable Data Types: Numbers In Python

Python supports integer, float and complex numbers defined as a class.




```
x=10 #Integer
y=10.1 #Float
z=1+2j #complex
```

edureka!

edureka!

edureka!

edureka!

Thank You

For more information please visit our website
www.edureka.co

edureka!