DATA TYPES

- Data Type represents the type of data present inside a variable.
- In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically.
- Hence Python is **dynamically Typed Language**

Python contains the following inbuilt data types

Text Type

```
1. **str** → Strings
```

Numeric Types

```
2. int → Integers
3. float → Floating point numbers
4. complex → Complex numbers
```

Sequence Types

```
5. list → Ordered, mutable collection
6. tuple → Ordered, immutable collection
7. range → Sequence of numbers
```

Mapping Type

```
8. dict → Key-value pairs
```

Set Types

```
9. set → Unordered collection of unique items
10. frozenset → Immutable set
```

Boolean Type

```
11. bool → True / False
```

Binary Types

```
12. bytes → Immutable byte sequences
13. bytearray → Mutable byte sequences
14. memoryview → Memory view of binary data.
```

15. None

---

1. int Data Type:
2. We can use int data type to represent whole numbers (integral values)
   Eg:

```
a = 10
type(a) #int
```

3. We can represent int values in the following ways
   - Decimal form
   - Binary form
   - Octal form
   - Hexa decimal form
4. Float Data Type:
   - We can use float data type to represent floating point values (decimal values)
     Eg:

```
f = 1.234
type(f) # float
```

   - We can also represent floating point values by using exponential form (Scientific Notation)
     Eg:

```
f = 1.2e3
# instead of 'e' we can use 'E'
```

```
print(f) # 1200.0
```

- The main advantage of exponential form is we can represent big values in less memory.

5. ***Note:*** We can represent int values in decimal, binary, octal and hexa decimal forms. But we can represent float values only by using decimal form.
6. Complex Data Type:
   - A complex number is of the form
     Eg:

```
3 + 5j
10 + 5.5j
0.5 + 0.1j
```

   - In the real part if we use int value then we can specify that either by decimal, octal, binary or hexa decimal form.
   - But imaginary part should be specified only by using decimal form.
7. bool Data Type:
   - We can use this data type to represent boolean values.
   - The only allowed values for this data type are: True and False
   - Internally Python represents True as 1 and False as 0

```
b = True
type(b) # bool

a = 10
b = 20
c = a<b
print(c) # True
```

8. str Data Type:
   - str represents String data type.
   - A String is a sequence of characters enclosed within single quotes or double quotes.
   - By using single quotes or double quotes we cannot represent multi line string literals.
   - For this requirement we should go for triple single quotes(''') or triple double quotes(""")
   - We can also use triple quotes to use single quote or double quote in our String.
   - We can embed one string in another string

```
s1="ashish soft"
 s1='''ashish soft'''

s1="""ashish soft"""

''' This is " character''' ' This i " Character '

'''This "Python class very helpful" for java students'''
```

Slicing of Strings

1. slice means a piece
   2. **Error! Bookmark not defined.**

```
s = "Ashish"

print(s[1])
print(s[1:22])
print(s[1:])
print(s[:4])
print(s[:])
print(s*3)
```

1. In Python the following data types are considered as Fundamental Data types
1. int
   2. float
   3. complex

4. bool
5. str

TYPE CASTING

- We can convert one type value to another type. This conversion is called Typecasting or Type coersion.
- The following are various inbuilt functions for type casting.

1. int()
2. float()
3. complex()
4. bool()
5. str()

int(): We can use this function to convert values from other types to int

```
int(123.987)  #123
int(10+5j) # TypeError: can't convert complex to int
int(True) # 1
int(False) # 0
int("10") # 10
int("10.5") # ValueError: invalid literal for int() with base 10: '10.5'
int("ten") # ValueError: invalid literal for int() with base 10: 'ten'
int("0B1111") # ValueError: invalid literal for int() with base 10: '0B1111'
```

> We can convert from any type to int except complex type.

float(): We can use float() function to convert other type values to float type

```
float(10) # 10.0 3)
float(10+5j) # TypeError: can't convert complex to float
float(True) # 1.0 7)
float(False) # 0.0 9)
float("10") #10.0
float("10.5") # 10.5
float("ten") # ValueError: could not convert string to float: 'ten'
float("0B1111") # ValueError: could not convert string to float: '0B1111
```

> We can convert any type value to float type except complex type.

bool(): We can use this function to convert other type values to bool type

```
bool(0) # False
2) bool(1) # True
3) bool(10) # True
4) bool(10.5) # True
5) bool(0.178) # True
6) bool(0.0) # False
7) bool(10-2j) # True
8) bool(0+1.5j) # True
9) bool(0+0j) # False
10) bool("True") # True
11) bool("False") # True
12) bool("") # False
```

str(): We can use this method to convert other type values to str type

```
str(10) # '10'
str(10.5) # '10.5'
str(10+5j) # '(10+5j)'
str(True) #'True'
```

Fundamental Data Types vs Immutability

- All Fundamental Data types are immutable. i.e once we creates an object,we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-chageable behaviour is called immutability.
- In Python if a new object is required, then PVM won't create object immediately. First it will check is any object available with the required content or not. If available then existing object will be reused. If it is not available then only a new object will be created. The advantage of this approach is memory utilization and performance will be improved.
- But the problem in this approach is, several references pointing to the same object, by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected. To prevent this immutability concept is required. According to this once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.

8. List Data Type:

If we want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.

- Insertion Order is preserved
- Heterogeneous Objects are allowed
- Duplicates are allowed
- Growable in nature
- Values should be enclosed within square brackets

```
list=[10,10.5,'ashish',True,10]
print(list) # [10,10.5,'ashish',True,10]
```

> list is growable in nature. i.e based on our requirement we can increase or decrease the
> Note: An ordered, mutable, heterogenous collection of eleemnts is nothing but list,where duplicates also allowed.

9. Tuple Data Type:

- tuple data type is exactly same as list data type except that it is immutable.i.e we cannot chage values.
- Tuple elements can be represented within parenthes

```
t=(10,20,30,40)
type(t) # <class 'tuple'>

t[0]=100 # TypeError: 'tuple' object does not support item assignment
t.append("ashish")
```

> Note: tuple is the read only version of list

10. Range Data Type:

- range Data Type represents a sequence of numbers.
- The elements present in range Data type are not modifiable. i.e range Data type is immutable.

11. set Data Type:

- If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.
- Insertion order is not preserved
- Duplicates are not allowed
- Heterogeneous objects are allowed
- Index concept is not applicable
- It is mutable collection
- Growable in nature

```
 s={100,0,10,200,10,'ashish'}
 s # {0, 100, 'ashish', 200, 10}
 s[0] # TypeError: 'set' object does not support indexing
```

- set is growable in nature, based on our requirement we can increase or decrease the size.

```
s.add(60)
s # {0, 100, 'ashish', 200, 10, 60}

s.remove(100)
```

```
s # {0, 'ashish', 200, 10, 60}
```

12. frozenset Data Type:

- It is exactly same as set except that it is immutable.
- Hence we cannot use add or remove functions.

```
s={10,20,30,40}

fs=frozenset(s) 3
type(fs) #  <class 'frozenset'>

fs # frozenset({40, 10, 20, 30})

for i in fs:print(i)
```

13. dict Data Type:

    If we want to represent a group of values as key-value pairs then we should go for dict data type.

```
d = {101:'ashish',102:'ravi',103:'shiva'}
```

- Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

```
d={101:'ashish',102:'ravi',103:'shiva'}

d[101]='sunny'
d # {101: 'sunny', 102: 'ravi', 103: 'shiva'}
# We can create empty dictionary as follows
d={ } # We can add key-value pairs as follows
d['a']='apple'
d['b']='banana'

print(d)
```

14. None Data Type:

- None means nothing or No value associated.