By Sai Kumar

PYTHON IDENTIFIERS – PART A

IDENTIFIERS

A Name in Python Program is called Identifier, also known as variables.

It can be Class Name OR Function Name OR Module Name OR Variable Name.

a = 10

Rules to define Identifiers in Python:

1. The only allowed characters in Python are

alphabet symbols (either lower case or upper case)

digits (0 to 9)

underscore symbol (_)

By mistake if we are using any other symbol like $ then we will get syntax error.

cash = 10 √

ca$h =20 x

2. Identifier should not start with digit

123total x

total123 √

3. Identifiers are case sensitive. Of course, Python language is case sensitive language.

total=10

TOTAL=999

print(total) #10

print(TOTAL) #999

By Sai Kumar

Identifier:

1) Alphabet Symbols (Either Upper case OR Lower case)

2) If Identifier is start with Underscore (_) then it indicates it is private.

3) Identifier should not start with Digits.

4) Identifiers are case sensitive.

5) We cannot use reserved words as identifiers

Eg: **def = 10 (x)**

6) There is no length limit for Python identifiers. But not recommended to use too

lengthy identifiers.

7) Dollar ($) Symbol is not allowed in Python.

Q) **Which of the following are valid Python identifiers?**

1) 123total

2) total123

3) java2share

4) ca$h

5) _abc_abc_

6) def

7) if

Note:

1) If identifier starts with _ symbol then it indicates that it is private

2) If identifier starts with __(Two Under Score Symbols) indicating that strongly private identifier.

3) If the identifier starts and ends with two underscore symbols then the identifier is language defined special name, which is also known as magic methods.

4) Ex**: __add__**

By Sai Kumar

RESERVED WORDS

In Python some words are reserved to represent some meaning or functionality.

Such types of words are called reserved words.

There are 33 reserved words available in Python.

True, False, None

and, or ,not,is

if, elif, else

while, for, break, continue, return, in, yield

try, except, finally, raise, assert

import, from, as, class, def, pass, global, nonlocal, lambda, del, with

Note:

1. All Reserved words in Python contain only alphabet symbols.

2. Except the following 3 reserved words, all contain only lowercase alphabet symbols.

True

False

None

Eg: a= true x

a=True √

>>> **import keyword**

>>> **keyword.kwlist**

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

By Sai Kumar

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

1) Int

2) Float

3) Complex

4) Bool

5) Str

6) Bytes

7) Bytearray

8) Range

9) List

10) Tuple

11) Set

12) Frozenset

13) Dict

14) None

Note: Python contains several inbuilt functions

1) type(): to check the type of variable

2) id(): to get address of object

3) print(): to print the value

By Sai Kumar

In Python everything is an **Object**.

1) int Data Type:

We can use int data type to represent whole numbers (integral values)

Eg: a = 10

type(a) #int

Note:

In Python2 we have long data type to represent very large integral values.

But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

We can represent int values in the following ways

1) Decimal form

2) Binary form

3) Octal form

4) Hexa decimal form

I) Decimal Form (Base-10):

It is the default number system in Python

The allowed digits are: 0 to 9

Eg: a =10

II) Binary Form (Base-2):

The allowed digits are: 0 & 1

Literal value should be prefixed with 0b or 0B

Eg: a = 0B1111

a = 0B123

a = b111

III) Octal Form (Base-8):

The allowed digits are : 0 to 7

Literal value should be prefixed with 0o or 0O.

Eg: **a = 0o123**

a = 0o786

IV) Hexa Decimal Form (Base-16):

The allowed digits are: 0 to 9, a-f (both lower and upper cases are allowed)

Literal value should be prefixed with 0x or 0X

Eg: **a = 0XFACE**

a = 0XBeef

a = 0XBeer

Note: Being a programmer, we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

a=10

b=0o10

c=0X10

d=0B10

print(a): 10

print(b): 8

print(c): 16

print(d): 2

Base Conversions

Python provide the following in-built functions for base conversions

I) **bin():**

We can use bin() to convert from any base to binary

1) >>> **bin(15)**

'0b1111'

2) >>> **bin(0o11)**

'0b1001'

3) >>> **bin(0X10)**

'0b10000'

II) **oct():**

We can use oct() to convert from any base to octal

1) >>> **oct(10)**

'0o12'

2) >>> **oct(0B1111)**

'0o17'

3) >>> **oct(0X123)**

'0o443'

III) **hex():**

We can use hex() to convert from any base to hexa decimal

1) >>> **hex(100)**

'0x64'

3) >>> **hex(0B111111)**

'0x3f'

5) >>> **hex(0o12345)**

'0x14e5'

2) **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): 12000.0

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

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.

4) **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

Eg:

a = 10

b = 20

c = a<b

print(c): True

True+True = 2

True-False = 1

5) **str Data Type:**

str represents **String** data type.

A String is a sequence of characters enclosed within single quotes or double quotes.

s1='sai'

s1="sai"

By using single quotes or double quotes we cannot represent multi line string literals.

s1="sai

kumar"

For this requirement we should go for triple single quotes(''') or triple double quotes(""")

s1='''sai

kumar'''

s1="""sai

kumar"""

We can also use triple quotes to use single quote or double quote in our String.

''' This is " character'''

' This i " Character '

We can embed one string in another string

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

Slicing of Strings:

1) slice means a piece

2) [ ] operator is called slice operator, which can be used to retrieve parts of String.

3) In Python Strings follows zero based index.

4) The index can be either +ve or -ve.

5) +ve index means forward direction from Left to Right

6) -ve index means backward direction from Right to Left

1) >>> **s="sai"**

2) >>> **s[0]**

's'

3) >>> **s[1]**

'a'

4) >>> **s[-1]**

'i'

5) >>> **s[40]**

IndexError: string index out of range

Try

s[1:40]

>>> **s*3**

>>> len(s)

Note:

1) In Python the following data types are considered as Fundamental Data types

int

float

complex

bool

str

2) In Python, we can represent char values also by using str type and explicitly char type is not available.

1) >>> **c='a'**

2) >>> **type(c)**

<class 'str'>

3) long Data Type is available in Python2 but not in Python3. In Python3 long values also we can represent by using int type only.

4) In Python we can present char Value also by using str Type and explicitly char Type is not available.

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

1) >>> **int(123.987)**

123

2) >>> **int(10+5j)**

TypeError: can't convert complex to int

3) >>> **int(True)**

1

4) >>> **int(False)**

0

5) >>> **int("10")**

10

6) >>> int("10.5")

ValueError: invalid literal for int() with base 10: '10.5'

7) >>> **int("ten")**

ValueError: invalid literal for int() with base 10: 'ten'

8) >>> **int("0B1111")**

ValueError: invalid literal for int() with base 10: '0B1111'

Note:

1) We can convert from any type to int except complex type.

2) If we want to convert str type to int type, compulsory str should contain only integral value and should be specified in base-10.

float():

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

1) >>> **float(10)**

10.0

2) >>> **float(10+5j)**

TypeError: can't convert complex to float

3) >>> **float(True)**

1.0

4) >>> **float(False)**

0.0

5) >>> **float("10")**

10.0

6) >>> **float("10.5")**

10.5

7) >>> **float("ten")**

ValueError: could not convert string to float: 'ten'

8) >>> **float("0B1111")**

ValueError: could not convert string to float: '0B1111'

Note:

1) We can convert any type value to float type except complex type.

2) Whenever we are trying to convert str type to float type compulsary str should be either integral or floating point literal and should be specified only in base-10.

bool():

We can use this function to convert other type values to bool type.

1) 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.

1) >>> **str(10)**

'10'

2) >>> **str(10.5)**

'10.5'

3) >>> **str(10+5j)**

'(10+5j)'

4) >>> **str(True)**

'True'

Fundamental Data Types vs Immutability:

All Fundamental Data types are immutable. i.e. once we create 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-changeable 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.

1) >>> **a=10**

2) >>> **b=10**

3) >>> **a is b**

True

4) >>> **id(a)**

1572353952

5) >>> **id(b)**

1572353952

bytes Data Type:

bytes data type represents a group of byte numbers just like an array.

1) x = [10,20,30,40]

2) b = bytes(x)

3) type(b) : bytes

4) print(b[0]) : 10

5) print(b[-1]) : 40

6) >>> for i in b : print(i)

10

20

30

40

Conclusion 1:
The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.

Conclusion 2:
Once we creates bytes data type value, we cannot change its values, otherwise we will get TypeError.

1) >>> **x=[10,20,30,40]**

2) >>> **b=bytes(x)**

3) >>> **b[0]=100**

TypeError: 'bytes' object does not support item assignment

bytearray Data Type:

bytearray is exactly same as bytes data type except that its elements can be modified.

Eg 1:

1) **x=[10,20,30,40]**

2) **b = bytearray(x)**

3) **for i in b : print(i)**

10

20

30

40

4) **b[0]=100**

5) **for i in b: print(i)**

100

20

30

40

Eg 2:

1) >>> **x =[10,256]**

2) >>> **b = bytearray(x)**

ValueError: byte must be in range(0, 256)

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.

1) Insertion Order is preserved

2) Heterogeneous Objects are allowed

3) Duplicates are allowed

4) Growable in nature

5) Values should be enclosed within square brackets.

Eg:

1) **list=[10,10.5,'sai',True,10]**

2) **print(list) # [10,10.5,'sai',True,10]**

Eg:

1) **list=[10,20,30,40]**

2) >>> **list[0]**

10

3) >>> **list[-1]**

40

4) >>> **list[1:3]**

[20, 30]

5) >>> **list[0]=100**

6) >>> **for i in list:print(i)**

...

100

20

30

40

list is growable in nature. i.e. based on our requirement we can increase or decrease the size.

1) >>> **list=[10,20,30]**

2) >>> **list.append("sai")**

3) >>> **list**

[10, 20, 30, 'sai']

4) >>> **list.remove(20)**

5) >>> **list**

[10, 30, 'sai']

6) >>> **list2=list*2**

7) >>> **list2**

[10, 30, 'sai', 10, 30, 'sai']

Note: An ordered, mutable, heterogenous collection of elements is nothing but list, where duplicates also allowed.