IDENTIFIER

A name given to variable, class, method.

```
x = 10  #int variable named x

def fun(): #function named fun
 ...

class Test(Exception): #class named Test
 ...
```

RULES TO DEFINE IDENTIFIER IN PYTHON

- Alphabet(A-Z, a-z) symbols, Digits(0-9)
- No special symbols except Underscore (_)
- Identifier should not start with numeric value
- Identifiers are case-sensitive
- No lenght limit for identifier name in python
- No keywords can be used

NOTE

- one underscore private variable
- two underscore strongly private
- two underscore followed by another two underscore language specific variable

RESERVED WORDS

Keywords are predefined, reserved and convey special meaning to the interpreter.
Total reserved words = 33

- True False None
- and or not is
- if else elif
- while for break continue return in yield
- try except finally raise assert

- import from as
- class def pass global nonlocal lamda with

DATATYPES

Represents the type of data stored in a variable.

- int
- float
- complex
- bool
- str
- bytes
- bytearray
- range
- list
- tuple
- set
- frozenset
- dict
- None

type() function tells the types of variable
id() to find the address of object in memory

--

int DATATYPE

Represents integral values

Types of values:

1. Decimal
2. Binary #starts with 0b or 0B
3. Octal #starts with 0o or 0O
4. Hexadecimal #starts with 0x or 0X

```
a = 56 #Decimal
b = 0b10011 #Binary
c = 0o773 #Octal
d = 0x1A5 #Hexa
```

Base Conversions

1. bin() #converts to binary value
2. oct() #converts to octal
3. hex() #converts to hexadecimal

--

float DATATYPE

Represents floating point values

Only decimal values are allowed.

```
a = 34.0
b = 6.92
```

Exonential Form

```
f = 1.2e3
print(f) #output : 1200
```

--

complex DATATYPE

c = a+bj a = Real Part
b = Imaginary Part
j = sqrt(-1)
c.img c.real

```
c = 4.6 + 5.1j #correct
c = 4 + 9i #incorrect
c = 5 + j7.4 #correct
```

--

bool DATATYPE

Represents two values for true or false
True False

Arithmetics

- True + True = True
- True + False = True

- False + False = False

--

str DATATYPE

Represents array of characters.
Written in 'single' or "double qoutes".
For multi-lines triple quotes
'''hello I am
Aditya'''

Slice Operator

Slice of a string is a substring.

```
[start:end:step]

s = "aditya"

s1 = s[0:3] #output : "adi"
s2 = s[2:4] #output : "it"
s3 = s[0:]  #output : "aditya"
s4 = s[3:]  #output : "tya"
s5 = s[:3]  #output : "adi"

# For Negative Index
# Negative index starts from -1 denoting the index of last element

s6 = s[-1] #output : "a"
s7 = s[-2] #output : "y"
s8 = s[-3:-1] #output : "ty"
s9 = s[::-1] #output : "aytida"
s10 = s[:]   #output : "aditya"
```

Printing string multiple times Multiply the string by the number you want to print it
s = "abc" s1 = s*3 print(s1) #will "abc" three timea

```
s = "abc"
print(s*3) #output : "abcabcabc"
```

.

Type Casting or Type Coersion

- int()
- float()
- complex()
- bool()
- str()

1. int() conversion

```
a = int(123.45)
print(a) #output : 123

a = int(12+1j) #ERROR : Type Error

a = int(True)
print(a) #output : 1

a = int("12")
print(a) #output : 12

a = int("12.4") #ERROR : only int type string
```

2. float() conversion

```
a = float(10)
print(a) #output : 10.0

a = float(9+7j) #ERROR : Type Error

a = float(True)
print(a) #output : 1

a = float("20.4")
print(a) #output : 20.4

a = float("abc") #ERROR : Type Eroor
```

3. complex() conversion

Form-1 : complex(x)==>x+0j
Form-2 : complex(x,y)==>x+yj

```
# complex() is an overloaded function

a = complex(10)
print(a) #output : 10+0j
```

```
a = complex(10.5)
print(a) #output : 10.5+0j

a = complex(False)
print(a) #output : 0+0j

a = complex("10.5)
print(a) #output : 10.5+0j

a = complex("two")
print(a) #ERROR : Type Error



a = complex(1,2)
print(a) #output : 1+2j

a = complex(2,2.4) #ERROR : Decimal imaginary value

a = complex("10","30") #ERROR
```

4. bool() conversion

```
bool(0)==>False
bool(1)==>True

bool(10+4j)==>True
bool(0+0j)==>False

bool("")==>False
bool("g")==>True
```

5. str() conversion

```
a = str(10)
print(a) #output : '10'

a = str(10.4)
print(a) #output : '10.4'

a = str(10+9j)
print(a) #output : '(10+9j)'

a = str('10abc')
print(a) #output : '10abc'
```

Python Object Refrenace

[Object Reference Tutorial]

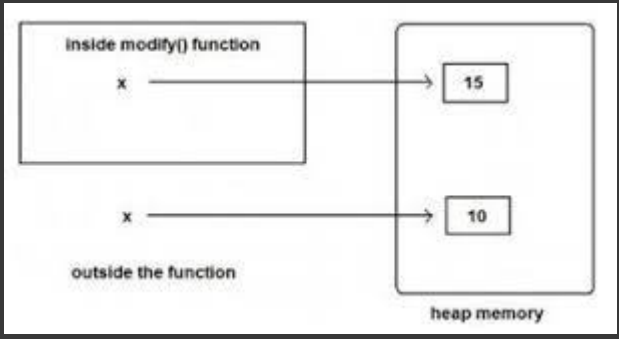
How Variables Are Created In Memory

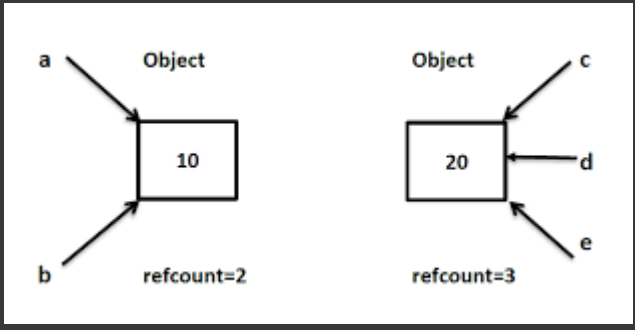
```
x = 10!
# An Obect 10 will be created in memory and x will point to that variable

x -----------------> 10

# Now if we replace value of x by 11 a new object 11 will be created
# and now x will point to 11

x -------↴               x10(Garbage Object)
         ↳---------> 11
```





How To Find Address Of Object

id() function gives the address of object in the memeory

- All fundamental data types are immutable

MORE ON DATATYPE

--

bytes DATATYPE

It reprsents a group of byte numbers just like an array

```
Syntax : bytes(src, enc, err)

Parameters:

src : The source object which has to be converted
enc : The encoding required in case object is a string
err : Way to handle error in case the string conversion fails.
Returns:  Byte immutable object consisting of unicode 0-256 characters according to src type.

integer : Returns array of size initialized to null
iterable : Returns array of iterable size with elements equal to iterable elements( 0-256 )
string : Returns the encoded string acc. to enc and if encoding fails, performs action according t
no arguments : Returns array of size 0.
```

python code to demonstrate int to bytes

```
number = 12
result = bytes(number)

print(result)  #Output : b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

--

bytearray DATATYPE

bytearray() method returns a bytearray object which is an array of given bytes. It gives a mutable sequence of integers in the range 0 <= x < 256.

```
Syntax: bytearray(source, encoding, errors)

Parameters:

source[optional]: Initializes the array of bytes
encoding[optional]: Encoding of the string
errors[optional]: Takes action when encoding fails
Returns: Returns an array of bytes of the given size.

source parameter can be used to initialize the array in few different ways. Let’s discuss each one
```

--

list DATATYPE

The list is a sequence data type which is used to store the collection of data. Tuples and String are other types of sequence data types.
Insertion order is preserved and duplicates are allowed.

- Order is preserved
- Duplicates are allowed
- Heterogenous datatypes are allowed
- Mutable
- Values should be enclosed with []

Creating A List

```
# Creating a List
List = []
print("Blank List: ")
print(List)

# Creating a List of numbers
List = [10, 20, 14]
print("\nList of numbers: ")
print(List)

# Creating a List of strings and accessing
# using index
List = ["Geeks", "For", "Geeks"]
print("\nList Items: ")
print(List[0])
print(List[2])
```

--

tuple DATATYPE

Tuple is a collection of Python objects much like a list. The sequence of values stored in a tuple can be of any type, and they are indexed by integers.

- Immutable
- Heterogenous
- Values should be enclosed with ()

Creating Tuples

```
Tuple1 = ()
print("Initial empty Tuple: ")
print(Tuple1)

Tuple1 = ('Geeks', 'For')
print("\nTuple with the use of String: ")
print(Tuple1)

list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))

Tuple1 = tuple('Geeks')
print("\nTuple with the use of function: ")
print(Tuple1)
```

Output

```
Initial empty Tuple:
()

Tuple with the use of String:
('Geeks', 'For')

Tuple using List:
(1, 2, 4, 5, 6)

Tuple with the use of function:
('G', 'e', 'e', 'k', 's')
```

--

range DATATYPE

Reprsents a sequence of values.
Immutable.

Form-1 : range(n) #Represents values from 0 to n-1

```
r = range(10)
type(r) #output : range
print(r) #output : range(0,10)
for i in r:
 print(i) #output : 0 to 9
```

Form-2 : range(start,end) #reprsents values from start to end-1

```
r = range(3,10)
type(r) #output : range
print(r) #output : range(3,10)
for i in r:
 print(i) #output : 3 to 9
```

Form-3 : range(start,end,step) #represents values from start to end-1 with gap of step value

```
r = range(3,10,2)
type(r) #output : range
print(r) #output : range(3,10,2)
for i in r:
 print(i) #output : 3 5 7 9
```

--

set & frozenset DATATYPE

A Set in Python programming is an unordered collection data type that is

- Iterable
- Mutable
- No duplicate elements
- Heterogenous
- No preserved order
- No indexing and slicing
- Set are represented by { } (values enclosed in curly braces)

frozensets are same as set but they are just immutable.
Has no attributes which changes the frozenset data variable.

Creating A Tuple

```
# a set cannot have duplicate values
myset = {"Geeks", "for", "Geeks"}
print(myset) #output : {'Geeks', 'for'}

# values of a set cannot be changed
myset[1] = "Hello"
print(myset) #output : TypeError: 'set' object does not support item assignment
```

--

dict DATATYPE

Dictionary in Python is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.
Dictionary holds key:value pair. Key-Value is provided in the dictionary to make it more optimized.

- No duplicate keys
- Can have duplicate values
- Mutable
- Dictionary keys are case sensitive the same name but different cases of Key will be treated distinctly.
- Dictionary are represented by { }
- Dictionary holds key:value pair. Key-Value is provided in the dictionary to make it more optimized.

Creating A Dictionary

```
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict) #output : {1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

--

None DATATYPE

The None keyword is used to define a null value, or no value at all.

None is not the same as 0, False, or an empty string. None is a data type of its own (NoneType) and only None can be None.

Creating A None

```
x = None
print(x) #output : None
```

Escape Sequence

Escape Sequence Meaning

- \' Single quote
- \\' Double quote
- \\ Backslash
- \n Newline
- \r Carriage Return
- \t Horizontal Tab
- \b Backspace

- \f Formfeed
- \v Vertical Tab
- \0 Null Character
- \N{Name} Unicode character Database named lookup
- \uxxxxxxxx Unicode character with a 16-bit hex value
- \Uxxxxxxxx Unicode character with a 32-bit hex value
- \000 Character with octal value ooo
- \xhh Character with hex value hh

Constants