Reserved Keyword

True, False, None, and, or, not, is, if, ~~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.

Data Types

Dynamically Type

→ In Python the type concept is available but we are not required to declare explicitly.

Data Types

- → int
- → float
- → complete
- → bool
- → str

} Fundamental Datatypes

- → list
- → tuple
- → set
- → frozenset
- → dict
- → bytes
- → bytearray
- → range
- → None

} Collection Related Data Types.

Everything in Python is object

```
a=10
type(a)
id(a) → to get the address
print(a)
```

In python long Data type Concept is avaliable but in Python 2 not in Python 3.

1. **int:**

```
a=10
```

→ decimal form (base-10)
→ binary form (base-2)
→ Octal form (base-8)
→ Hexa form (base-16

(i) decimal (base-10)
0 to 9

```
a=123
```

(ii) binary (base-2) ← 0B / 0b
0 & 1

```
a=0b1111
print(a) ⇒ 15
```

(iii) Octal (base-8)
0 to 7

→ 0o or 0O (zero Small o or zero capital O)

```
a=0o123
print(a) ⇒ 83
```

Slice operator:

s = 'abcdefghijklmnopqrstuvwxyz'

→ s[begin : end]

→ return substring (slice) from begin index to (end-1) index

print(s[3:9])

⇒ defghi

default begin → 0

default end → end of the string.

s[3:1000] → this slice operator never going to raise index error. it will consider till the end of ~~er~~ string.

[5:1] ⇒ empty.

+ operator

s = 'durga' + 'soft'

print(s) ⇒ durgasoft

s = 'durga' + 10

↳ print(s) ✗

Type error both should be string.

both the arguments must be string type.

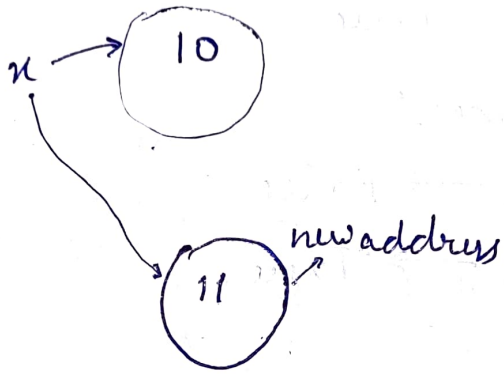
*Fundamental Data Types Vs Immutability

mutable → changeable

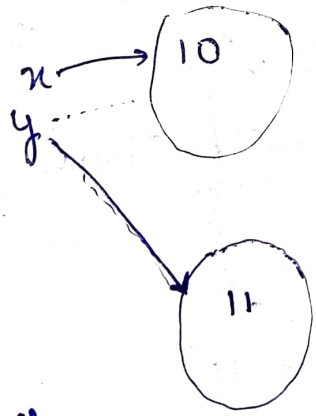
Immutable → Non Changeable

→ Once we create an object, we cannot perform any changes in that object. If we are trying to perform any changes, with those changes, a new object will be created.

```
x = 10
print(id(x))
x = x+1
print(id(x))
```

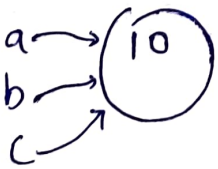



```
x = 10
y = x
print(id(x))
print(id(y))
y = y+1
```



→ Object Reusability ***

```
a = 10
b = 10
c = 10
```




```
print(id(a))
print(id(b))
print(id(c))
```
} same

Memory utilization } benifits
Performance ↑

Collection Related Data Types

List
tuple
set
frozenset
dict
range
bytes
bytearray

} these collection Related Data types can hold multiple values.

1. List :

→ If you want to represent a group a values or object as a single entity.

→ Duplicates are allowed

→ Order is Important (preserved)

→ l = [10, 20, 30] → list
l = [10 "durg", 20, 10, 30]

→ Hetrogenous objects are allowed

print(l[0] → 10
print(l[-1] → 30
print(l[1:4] → durga, 20, 10

→ Indexing Applicable
→ slicing "

→ l.append(10)

→ list can be appended by append function

→ l.remove(30) → remove element

(i) Order preserved
(ii) duplicate objects are allowed
(iii) [ ]
(iv) Heterogenous objects
(v) Indexing & Slicing
(vi) Growable in nature
(vii) Mutable

2. Tuple :

→ Exactly same as List except that it is **immutable**

→ Read only version of List is Tuple

t = (10, 20, 30, 10, 'durg')

→ print (t[0]) → 10

→ print (t[-1]) → durg

→ print (t[1:4]) (20, 30, 10)

→ print (t) → (10, 20, 30, 10, 'durga')

→ * t[0] = 7777
↳ Type Error → As tuple is immutable

t.append(30) } X Attribute Error
t.remove(10) } tuple objes has no attribute 'append'

→ * t = ( ) → empty tuple

→ * t = (10) → this is not tuple, it is int type.

→ * t = (10,) → now this will be considered as tuple.

→ * Order is preserved

List	Tuple
① mutable	① Immutable
② []	② ()
③ more memory	③ Less memory
④ Performance is less	④ Faster Access → performance is more (as content is find).

e.g:

3. Set

* duplicates are not allowed
* Order not required

```
s = {10, 20, 30, 40}
s = {10, 20, 10, 'durg', 30, 40}
print(s) ⟶ {40, 10, 'durga', 20, 30}
           order will not be preserved
print(s[0]) ⟶ Type Error: Index 'set object does not support indexing.
```

* Indexing / Slicing not applicable
* Heterogenous objects are allowed

⟶

```
s = {10, 20, 30, 40}
s.add(50) ⟶ {40, 10, 50, 20, 30}
print(s)
s.remove(30)
print(s) ⟶ {40, 10, 50, 20}
```

* Growable & Mutable

Append() vs add()

l = [10, 20, 30, 40] ← 50

l.append(50)

because this element is always going to be added at last

[10, 20, 30, 40, 50]

s = {10, 20, 30, 40}

s.add(50)

but here there is no garantee that this element is going to be added at last.

* s = { } → this is not set but empty dict.

s = set() → this how empty set is created by set() function

list	set
① order ✓	① order ✗
② duplicates ✓	② duplicates ✗
③ []	③ { }

4. Frozen Set

```
s = {10, 20, 30, 40}
fs = frozenset(s)
print(type(fs))
fs.add(50)
fs.remove(30)
```

✗ AttributeError: 'frozenset' object has no attribute 'add'

⟶ Immutable

Tuple	Frozenset
① Order preserved	① order not applicable
② duplicates are allowed	② duplicates are not allowed
③ index, slice applicable	③ Index, slice not applicable.

5. dict:

⟶ key value pairs

d = {K1 : V1, K2 : V2, K3 : V3}

d = {100 : 'durga', 200 : 'sunny' 300 'chinny'}

d = { } ⟶ Empty dictonary

d[key] = value

d[100] = 'durga'

→ hash order is not imp.
→ duplicate keys are not allowed.
→ If we add key again the value will be updated.
→ *Order is not preserved
→ Hetrogenous objects allowed
→ Mutable
→ Indexing, Slicing not allowed.

6. Range:

→ If you want to represent a sequence of numbers, then go for range Datatype.

```
r = range(10)  # 0 to 9
print(r) ⟶ range(0,10)

for x in r:
    print(x) ⟶ 0
               1
               .
               .
               9
```

1 ⟶ range(n)
from 0 to n-1

2 ⟶ range(begin, end)
begin to end-1

3 ⟶ range(begin, end, increment/decrement)

```
r = range(1, 21, 2)
  1, 3, 5, 7, 9 ....
r = range(20, 1, -5) # 20, 15, 10, 5, ..
```

→ Order

→ Indexing / Sclicing applicable

```
r = range(10, 201)
print(r[0]) → 10
print(r[-1] → 20

r1 = r[1:5]
for x in r1:
    print(x) → 11, 12, 13, 14
```

→ Immutable

If you try to change

```
r = range(10,21)
r[1] = 1000
```

TypeError → Range Object does not support item assignment

7. bytes data type

```
l = [10, 20, 30, 40]
b = bytes(l)
print(type(b)) → <class 'bytes'>
for x in b:
    print(x)
```

10
20
30
40

→ We we want to handle binary data like images, video files, audio file, then we should go for bytes data type & byte array data types.

1. **break**

based on some condition if we want to break loop execution.

```
for i in range(10):
    if i==7:
        print("processing is enough... plz break")
        break
    print(i)
```

```
0
1
2
3
4
5
6
processing is enough... plz break
```

} o/p.

2. **Continue**

```
While condition:
    body
    continue ⟶ skip current and continue for next iteration
    body
```

```
for i in range(10):
    if i%2==0:
        continue
    print(i)
```

```
1
3
5
7
9
```

} o/p

break else.

```
cart=[10,20,60,70,90]
for item in cart
    if item>500:
        print("Sorry we cannot process this item - ",item,
              "insurance must be required")
        break
    print("Processing item:
else:
print("Congrats"): executes when break statement
                   does not execute through the loop
```

else → here means loop without break.

How to exit from loop
↳ break

How to skip current iteration and continue for the next iteration
↳ continue

when else part will be executed
→ if loop executed without break.

3 pass statement

```
if true:
    pass
else:
    print("Hello")
```

→ It is an ~~expt~~ empty statemt
→ It won't do anything

```
def f1():
    print("Hello")

def f2():
    pass
```

4. del statement

key word in python

```
x = 10
print(x)
del (x)
```

del and None

del x → if we want to delete variable

x = None → when we want variable but don't want the object then assign with None.

List

→ insertion order is preserved and duplicates are allowed
→ Hetrogeneous
→ Growable
→ l=[] → Empty
l= [10,20,30,40)
l= eval(input("Enter some list")) form CLI
l= list(sequence)
l= ~~s.splitter~~ s.split(seperator)

Nested Lists

→ [10,20,[30,40]]
we can access by index, slice

Traversing the elements of List:

i) By using while loop.
ii) By using for loop:

Important function and Methods of List:

1. len(list)
2. count(x)
   l=[10,20,30,40,10,20,10,10]
   print(l.count(10)) → 4
3) index() → returns first occurence

Manipulating elements of Lists:

→ l.append(element)

2→ insert(index, element)
- if index greater than last index then last index
- if val index smaller than first then at the beginning

append() vs insert

↓ adds element to the end

↘ adds the element at specified position.

3. extend()
   ↳ adds one list to other

4) remove() and pop(index)

5) reverse()

6) sort()

{+, & * operator can be applied to the list}

Comparing List Object.

```
x = ['Dog', 'Cat', 'Rat']
y = ['Dog', 'Cat', Rat']
z = ['DOG', 'CAT', RAT']
print(x==y) # True
print(x==z) # False
print(x!=z) # True
```

1) The number of element must be equal
2) The order should be same
3) The content should be same (including case)

```
x=[10,20,30],[40,50,60],[70,80,90]
print(x)
print("Elements Row wise:")
for r in x:
    print(x) ⟶ [10,20,30]
               [40,50,60]
               [70,80,90]
print("Element in Matrix style:")
for i in range(len(x)):
    for j in range(len(x[i])):
        print(x[i][j],end=' ')
    print()
```

```
10 20 30
40 50 60   } o/p
70 80 90
```

List Comprehensions

```
l1 = [x * x for x in range(1,11)]
print(l1) ⟶ [1, 2, 4, 25, 36, 49, 64, 81 100]
```

⟶ list [expression for x in sequence]

⟶ list=[expression for x in sequence if condition]

Tuple

```
t=()
t=(10,)
t=(10,20,30)
t= tuple(sequence)
```

Access elements of tuple:

⟶ index or slice

⟶ operator ⟶ +, *

Important function of tuple

1. len() :
   t = (10, 20, 30, 40)
   print(len(t)) → 4

2. Count()
   t.count(10) → 1

3 index()
   t.index(10) → 0 index of first occurrence.

4. t.index(90) → Value Error

5. sorted()
   To sort elements based on natural sorting order.
   t = (30, 10, 50, 40, 20)
   print(t.sorted()) → [10, 20, 30, 40, 50]
   t.sort() → tuple object has no attribute 'sort' reason tuple is immutable.

~~print(tuple(sorted~~
   t2 = tuple(sorted(t, reverse = True))
   print(t2) → (50, 40, 30, 20, 10)

6. min() → min(t) } For Hetrogeneous
7. max() → max(t) } this won't work.

Operator ~~happily~~ happily applicable for Tuple as List (same).

Tuple Packing and Tuple unpacking

packing ⟶ Grouping into single.

```
a=10
b=20
c=30
d=40
t= a,b,c,d
print(t) ⟶ (10,20,30,40)
```

Unpacking

```
t= (10,20,30,40)
a,b,c,d=t
```

Tuple Comprehension:

```
t= (x * x for in range(1,11)
   print(type(t)) ⟶ Generator
```

So tuple Comprehension not supported in Python.

Difference b/w List and Tuple

1. List is a group of comma seperated values with in square [] brackets.
[] Square brackets are mandatory.

1. tuple is a comma seperated values with paran parentheses () and () are optional. eg t= (10,20,30,40)
t= 10,20,30,40

2.

2. List are mutable

2. Tuple objects are immutable i.e once we create

3 List object require more memory

3. Tuple object require less memory

4. In the case of list every time new object are created.
List object won't be reusable.

4. Tuple object can be reused.

5. List → Spead is slow relatively
5 tuple → spead i.e performance is high less time

6. List → Comprehension concept is applicable
tuple → Comprehension concept is not applicable.

7. List objects are hashable type and hence we cannot store an

~~7.~~

7.

8. Content keeps on changing go for list
8. Content is constant then go for tuple

9. Unpacking is possible but packing is not possible in list
9. Both packing & unpacking possible in tuple.

Unpacking
```
l = [10, 20, 30, 40]
a, b, c, d = l
```

packing
```
a = 10
b = 20
c = 30
d = 40
t = a, b, c, d
```