Python Datatypes

- Numbers
- List
- Tuple
- String
- Set
- Dictionary
- Conversion

Everything is an object in Python
∴
Datatypes are actually classes
Variables are instance of class

type( ); returns value to which class datatype variable belongs

isinstance(,); returns value if object belongs to particular class

* Numbers

Integers, floating point, complex Number

```
a = 5          # integer ← type(a), isinstance(a, integer);
b = 3.14       # Float                          ↓
c = 1 + 2j     # complex                       int
```

c.imag returns imaginary value
c.real returns real value

* List []

- Ordered sequence of items
- all the items in a list do not need to be of same type
- Defined with [] parentheses separated by commas

```
a = [1, 2.2, 'python']
    0   1     2
```

a[2] = 'python'
a[0:2] = [1, 2.2]
↗ 0 to 1
a[1:] = 'python'
after 1 →

- List are mutable (value can be changed)

a[1] = 3.14 # possible

* Tuple()
  - Ordered sequence of items
  - Tuple are immutable (i.e dynamically not changeable)
  - Faster than list (because not mutable)
  - Defined by () parenthesis separated by commas

```
          0      1        2
t = ( 5, 'program', 1+2j)

t[1] = program

t[0:2] = (5, 'program')
```

Slicing t[0:n] means 0 to n-1 index
t[n:] means after n index

* Strings
  - Sequence of Unicode characters
  - Represented by single quotes or double quotes
  - Multiline strings denoted by triple quotes """abc""" or ''' ''' '''abc'''
  - Strings are immutable

```
s = ' Hello World '      s[0]='F' #not possible
    0 1 2 3 4 5 6 7 8 9 10
```

* Set {}
  - Unordered collection of unique items
  - Defined by ~~()~~ { } parenthesis separated by commas
  - Set operation like Union, Intersection can perform
  - Repeated values are eliminated because it is collection of unique items
  - Do no support Indexing s[1] = X

```
S = { 1, 3, 2, 7 }
```

* Dictionary

- Unordered collection of key-value pair
- They are optimized for retrieving data we must know the key to retrieve value
- Defined by {} with each item being in a pair 'key': 'value'

```
d = { 'key1': 'value1', 2:4 }

d['key1'] = value1
d[2]      = 4
```

* Conversion between datatype

Conversion Function	ip → op
int()	Float → int , string → int ('5' → 5)
float()	int → float , string → float ('3.1' → 3.1)
str()	int → string (5 → '5')
	Float → string (3.1 → '3.1')
set()	list → set
tuple()	set → tuple
list()	string → list
dict()	Example
	dict ([(3,9) , (4,16)])
	key value

Type conversion

- Implicit type conversion
- Explicit type conversion

* Implicit type Conversion
  Python automatically convert one datatype to another, by promoting conversion of lower datatype to higher datatype.

* Explicit type Conversion

  int(), float(), str(), etc

Typecasting
  Syntax
    (required DataType) (expression)

- Type conversion is conversion of object from one datatype to another
- Implicit is done by Interpreter
  It avoids loss of data
- Explicit → loss of data because we Force to convert

Input, Output & Import

* output

actual syntax

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

sep=' ' - default space character
end='\n' - default newline
any changes to sep='#' assign &
end='&' assigned would reflect to screen

- str.format()
  {} are place holder

a=5; b=10
print('Value of a = {} & b = {}'.format(a,b))
op:- Value of a = 5 & b = 10

{labelmeaning will replace by value}

- % operator.

pi = 3.14159265358979323844

print('Correct upto 4 decimal value of pi is 1.4f' %pi)
op:- Correct upto 4 decimal value of pi is 3.1415

* Input

Syntax:-

var = input([prompt])

Entered value would be string
we need to convert in appropriate

* Import

Syntax:- # file.py to be import
import file

mostly constant all are saved in
constant.py file and imported.

Operator

+	>	and	&	=	is
-	<	or	|	+=	is not
*	==	not	~	-=	in
/	!=		^	*=	not in
%	>=		>>	/=	
//	<=		<<	%=	
**				//=	
				**=	
				&=	
				|=	
				^=	
				>>=	
				<<=	

* Name

What is name in python/identifier in py

Everything in python are objects

simply name is given to object to access them

Variable, function are object

a = 2

2 is object stored in memory

a is name of object

id(a) or id(2) → gives the memory location where object 2 is stored

* Namespace

namespace is collection of names

mapping

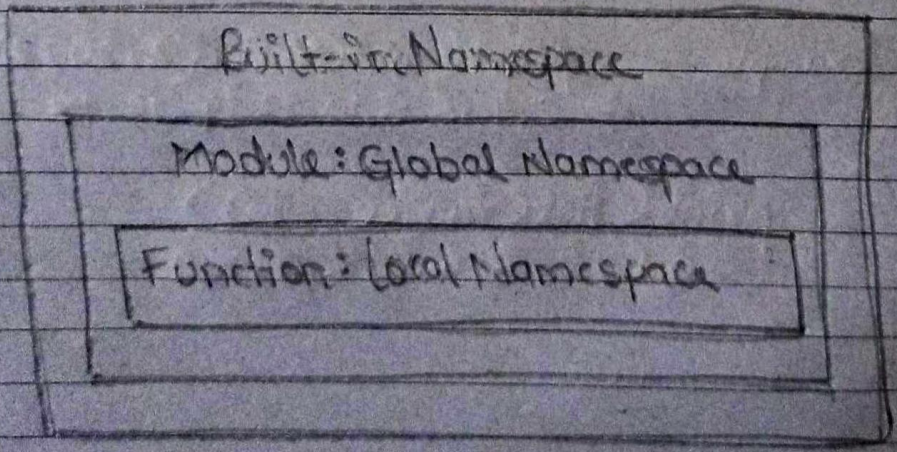
- namespace containing all built-in names is created & started by Interpreter till we exit

The reason why we do not need to import builtin fun name #include<iostream> for cin, cout std::

using namespace

Each module create its own global namespace

- Different namespace are isolated Hence same name that exist in different module do not collide
- A local namespace is created when function is called



* Variable Scope

Scope is portion of program from where namespace can be accessed directly without prefix

3 nested loop

Scope of Current fun with local name
Scope of module with global name
Outmost scope which as builtin name

if ref mode inside fun
name is searched in
localscope then global the built in namespace

fun1 → local
fun2 → newscope nested inside localscope

use of global keyword defines scope of object as global

* if Else

Syntax
```
if test-Expression :
    □
```
indentation
unindentation

body of if is inside indented section
1st unindentated line marks end

Syntax
```
if test-Expression:
    □
else:
    □
```

Syntax
```
if test-Expression:
    □
elif test-Expression:
    □
else:
    □
```

Nested if

* For Loop

list, tuple, string

Syntax :-

```
for val in sequence:
    Body of for
```

val takes value of item inside seq for each iteration
Loops continues until last item is reached

* Range function

range(10) → generate number from 0 to 9

range(start, stop, step)
- This function do not store all values
- It remember & generate next value to go

To force function to output list function is used

list(range( ))

Example print(list(range(2,20,2))
→ 2,4,6,8,10,12,14,16,18

print(list(range(0,10))
→ 0,1,2,3,4,5,6,7,8,9

* For else loop

Syntax :-

```
For val in sequence:
    Body of for
else:
    Body of else
```

else block executed if items in for exhausts
break in for do not execute else too

* While Loop

Syntax:-

```
while test-expression:
    Body of while
```

* While else loop

Syntax:-

```
while test-expression:
    Body of while
else:
    Body of else
```

* break Statement

It terminate the loop inside which
break statement is present
Another block after this loop end is executed

* continue statement

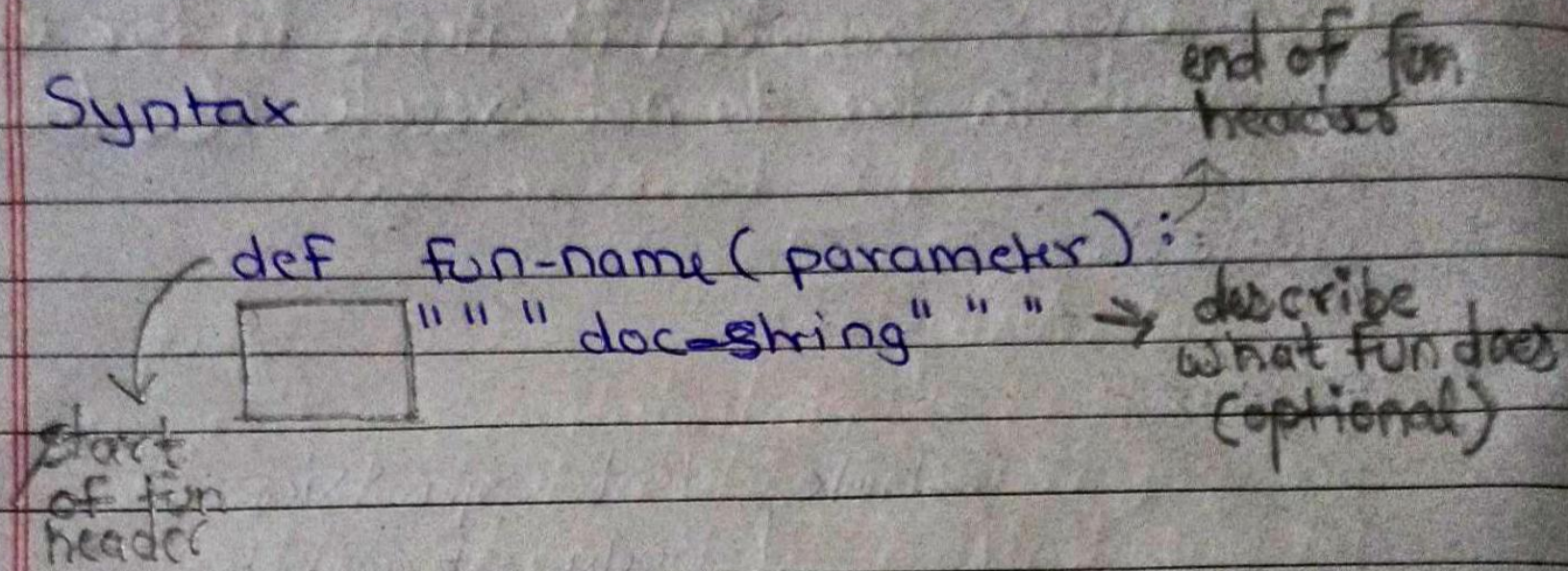
It iterate loop again for loop containing
this statement

* pass statement

It is null statement,
difference between comment & pass
pass do not terminate / ignore like
comment ignored by interpreter

* Functions

Syntax



fun-name( ) # fun Call

print(fun-name.__doc__)
prints docstring in triple quotes