|  |
| --- |
|  |

**Python**

Modern very high-level language that has all the good bits of js but still keeps a type system intact. Interpreted language, meaning no compilation and linking is necessary before the code is ran. ‘;’ aren’t used in python, however proper indentation is a must.

• Arguments: Any arguments passed to the python interpreter are available in the sys module’s argv list for a python program. No args means argv has size 1 and the argv[0] is “”.

• Character encoding used by the interpreter for interpreting a document: Use this to change the default recognized encoding from UTF8 to others. This line must be the first line (although after #! <env location> line if present).

# -\*- coding: <encoding> -\*-

• Comments: # and “”” “””.

• Data Types:

String: str(), has ‘’ , “”, ‘’’ ‘’’ or “”” “””, if we multiply a number with a string, then the string is returned number times. Strings inside () are joined together, so (‘abc’ ‘cde’) become ‘abcde’, doesn’t work with literals. “{<replacement field>}”.Format(<args>), inserts the arg’s value at the field. There can be multiple replacement fields and they can either have an index or a name.

“hi {name}”.format(name=”yo”) works.

“{} {}”.format(“as”,”bb”) Here first {} is 0 and next is 1 automatically.

“{0.wing} “.format(bird) will call bird.weight.

“{0!s} {name!r} {!a}”.format(“yoo”,name=”gama”,”aa”): will call str() on 0 before passing it to replacer for yoo, repr() for name and ascii() for aa.

Strings are immutable.

List: list(), or can be declared with []. Lists in python can have any data type. To shallow copy a list we can use slicing.

Lists support +, i.e., 2+ [2,3,4] makes it 2,3,4,2. Lists are mutable.

List1[:]=[] copies empty list into List1.

• Slicing: Any list data type, even strings can be sliced. Slicing means that only a part of the list is returned.

Syntax:

[<start index(inclusive)>:<end index(exclusive)>:<stops>]

aList=[1,2,3,4][1:2:1] returns 1,2, [2:1:-1] returns 2,1, [::2] returns 1,3 (empty means 0 for start, length-1 for end and 1 for stops)

• len(<iterable>): Get length of iterable.

• If..elif..else is followed here.

• For: for x in y, for x,y in z and for i in range(<start>,<stop>,<step>)

• break, continue and else: Break and continue are like other languages but else block for a loop works after the loop has finished iterating and has not encountered a break.

• pass: skip the current block.

• Functions: def <funcname>(<params>):. Params don’t need types. The first line of a function can be a string, it is known as docstring. The execution of a function is same as in c++, functions create their own local symbol table where variables are stored. If they are referenced then first the local symbol table is searched, then the symbol table of the enclosing scope and so on until global symbol table is searched and lastly built-in names symbol table.

We cannot hide variables in python, so if a higher table declares a variable then the same name variable can’t exist in local scope.

Since python is an interpreted language, functions can only be called after they are declared in the file.

When we define a function its address is automatically inserted in the local symbol table of enclosing scope. So we can simply pass the function name to pass function as reference.

Every function returns something, if return isn’t defined then ‘None’ is returned.

def func(a,b=2): Default value for params.

i=2

def func2(args=i):

…

i=3

func2() will pass 2 to func. The default values are taken at the time of definition of a function and so they don’t change. They are taken only once, this means reference types like list,dict or classes work on the same instance.

For ex:

def func3(l=[]):

l.append(2)

return l

func3()

func3()

will return [2] and [2,3], to avoid this use l=None and then check if l is None and assign new reference type.

Special parameters:

By default, normal parameters can be passed either by position or keyword (kwargs- keyword arguments) but theres also pos only and kw only params. In a function call and definition the positional arguments are specified before keyword arguments.

Pos only:

def func(<arg1>,/): Can only be invoked with func(<val>).

Kw only:

def func(\*,<arg>): Can only be invoked with func(<arg>=<val>)

\*:

def func(\*args): Can receive any number of elements and it will create a tuple.

\*\*:

def func(\*\*args): Can receive dictionary and it will have <key>=<value> pairs.

func(a=2,b=3,c=4,hello=5) will pass a,b,c,hello as keys and their values as their values in the dict.

\* must occur before \*\*. And \* must occur after formal parameter list.

Formal parameter names can’t be directly passed to \*\* dictionary, i.e.,

for def func(name,\*\*args):

func(name=2,\*\*{‘name’:2}) will fail.

To resolve this we use positional only args

def func(name,/,\*\*args): will work for that call.

We can get docstring of a func using, func.\_\_doc\_\_ .

Annotations:

Function definitions may specify types of the params. They are then stored in the func.\_\_annotations\_\_ dictionary.

def <funcname>(<arg>[: <type>=[<default value>]]):

• Python supports chain assignment (a=b=c) and multiple assignments (a,b=c,d)