**Language Description:**

* Expecting language is a mixture of python(simple syntax) and c++(independency)
* Implementing in c++
* Python like structure i.e. no main func
* OOP paradigm
* Static strong typing with implicit up typecasting
* All keywords are start with small letters except True/False/None

**KEY WORDS:**

1. int/char/float
2. Boolean
3. string
4. self
5. super
6. is

TrueisTrue / FalseisFalse / NoneisNone ->true

[]is[] / {}is{} ->false

''is'' ->true

1. True/1, False/0
2. None
3. break
4. continue
5. class

class ExampleClass:

1. def

def acessModifer FunctionName(): ???

1. del

a = b =5

del a

print(a)

* name 'a'isnotdefined

a =['x','y','z']

del a[1]

a

* ['x','z']

1. if
2. else
3. elif

if a ==1

print('One')

elif a ==2

print('Two')

else

print('Something else')

1. try
2. except
3. raise – to raise your own exception

raiseZeroDivisionError('cannot divide')

1. finally

try

Try-block

excerpt exception1

Exception1-block

excerpt exception2

Exception2-block

finally

Finally-block

1. for

foriin names:

print('Hello '+i)

1. in

a =[1,2,3,4,5]

5 in a ->true

10in a ->false

1. global

globvar=10

def read1():

print(globvar)

def write1():

global globvar

globvar=5

def write2():

globvar=1

read1() 10

write1()

read1() 5

write2()

read1() 5

1. lambda

a =lambda x: x\*2

foriin range(1,6):

print(a(i))

**Output** 2 4 6 8 10

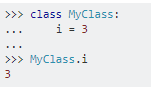
1. return
2. while

while condition:

//do something

1. public (varName), private (\_\_varName), protected (\_varName)
2. @ (static)

@varName???



def@funcName(parameters)

1. +/$ (final)???

+varName

1. \* (pointer)
2. assert   
   //for debugging purpose
3. pass

**OPERATORS:**

*UNARY :*

1. ++, --
2. !, ~

*BINARY:*

1. \* / %
2. + -
3. <<= != >>= ==
4. &&
5. =
6. ||
7. |
8. &
9. \*= /= += -= %=
10. ? :
11. ^(exp)

**PUNCTUATORS:**

1. + Concatenation
2. , separate elements of list and dictionary
3. : used in dictionary
4. (
5. )
6. [
7. ]
8. {
9. }

**IDENTIFIERS:**

alph(alph+digit+\_)\*

**CONSTANTS:**

*INTEGERS:*

(+ + - + ^)digit\*

*FLOAT:*

(+ + - + ^)digit\*(.)digit+(^+((E+e).(+ + - + ^)digit+)

*CHAR:*

‘( \ (A + B + ‘ + \) + B + C)’

*STRING:*

“( \ (A + B + “ + \) + B + C)\*”

**SYNTAX:**

1. List / array

my\_list=[1,"Hello",3.4]

my\_list = ["mouse", [8, 4, 6], ['a']]; //list inside list

1. Dictionary

my\_dict = {'name':'Jack', 'age': 26}

print(my\_dict['name']) -> Jack

people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},

2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}} //nested dictionary

1. Constructor  
   def \_\_init\_\_(self):  
   self.integer = 1

self.char = ‘1’

1. Inheritance

class Penguin[Bird, Animal] //bird, animal=parent, penguin=child

**Classification Table**

|  |  |
| --- | --- |
| Class Name | member |
| Boolean | True  False  0  1 |
| Datatype | int  char  float |
| String | string |
| AccessModifier | i.epublicIndentifier  \_ i.e \_protectedIdentifier  \_\_ i.e\_\_privateIndentifier |
| Static | @ i.edef @identifier(arg): |
| Final/Const | $ i.e $identifier = 1 |
| If | if |
| Else | else |
| Elif | elif |
| For | for |
| While | while |
| Continue | continue |
| Break | break |
| Def | def |
| Return | return |
| None | None |
| Pass | pass |
| Raise | raise |
| Assert | assert |
| In | in |
| Try | try |
| Except | except |
| Finally | finally |
| Del | del |
| Global | global |
| Lambda | lambda |
| ====================== | ===================== |
| Self | self/this/indentifier |
| Super | super |
| Class | class |
| New | new |
| ========Operator===== |  |
| Not | !  Not  ~ ??? |
| Pointer  Dereference | \* |
| Multiply | \* |
| DM | /  % |
| PM | +  - |
| RelOp | Is  ==  !=  <=  >=  <  > |
| AsOp | +=  -=  \*=  /=  %= |
| And | and |
| Or | or |
| BitOr | | |
| BitAnd valueOf  refOf | & |
| = | = |
| ^ | ^ |
| : | : |
| ? | ? |
| AcOp | .  -> |
| =======punctuator======= |  |
| Concat | + |
| : | : |
| , | , |
| ( | ( |
| ) | ) |
| [ | [ |
| ] | ] |
| { | { |
| } | } |
|  |  |
|  |  |