3/1/23

====================================

Python Keywords :

Keywords areunique or reserved words which are pre defined meaning.

C ==> 32/48

py==>35

ex:-True, False, None,assert, if , elif.....

#.....p10

===========

#prog to demonstrate keywords

a=(True==1)

print(a)

b=a+10

print(b)

c=(False==199)

print(c)

d=c+10

print(d)

===========

python Literals:

are classified into 4 categories

String L

when evr we assign a string data to a var or const -->SL

s1="xyz"

types of string:

single line string - s1='ccc'

multi-'aaa

bbb'

Numeric L:

when evr we assign a numeric data to a var or const -->NL

a:int a=10

b:float b=1.1

c:complex: c=2+3j

Boolean L:

when ever we assign a boolean data to a variable o\r const --> BL

a=True

b=False

print( )

Special L

when ever we assign a None data to a variable or const Sp.L

a=None

p11.py==>prog to demonstareat Literals

=====================

LAB

====================

Python Ops:

they are symbols which performs a specific operation on operands

Ex: a+b, a\*c, x/y, (a+b)\*d............

+, \_. \*, /===>operatoers

ag,d,f,h,===>operands

In general Op are classified into 3 Cat

1)Unary

Any operation--Single operand

ex: a++

2)Binary

Any operation--Two operand

ex: a+b

3)Ternary

Any operation--Three or more operand

ex: (a+b)\*c

Python Operators:

1)Arithmetic op:

+, -, \*, /, %, //

2)relational:

< > <= >= != ==

3)Logical:

&& || !

4)Bitwise:

& | ^ ~ << >>

5)Assignment:

=

Note:

+= -= \*= /= %= //=

6)membership :

in : if 1st found in 2nd returns true

not in :if 1st not found in 2nd returns true

7)Identity

is : if 1st and 2nd points to same Obj returns true

is not :if 1sst and 2nd points to diff Obj returns true

====================

p12.py ==>arth op --> //

p13.py===Rel !=

p14======Log ||

p15 bitwise ~

p16 Arth assignment %=

p17 membership not in

p18 identity is

======================

python control structure statements:

CS statements are the one which alters/change the flow of execution of a program

1)py Dec making statements ---> to skip

>if

>if else

>nested if

>elif

2) python loops---->to repeat

>for

>while

>do-while

> optional for each

python DMS

> if

syntax:

if cond:

statements

========================================================

>if else

syntax:

if cond:

statements

else :

statements

ex: p20.py

a=10

if a%2==0:

print("Even no")

else :

print("odd no")

>nested if

Syntax :

if cond :

if cond :

if cond :

statements ====> it executes only if all the three conditions are true

EX : p21.py

a=18

if a<10:

if a>5:

print("a is less than 10 and greater than 5")

else :

print("inner if not executed")

else:

print("outer if not executed")

>elif

if cond:

statement

elif cond:

statement

elif cond:

statement

else:

statement

ex p22.py

a=7

if a<5:

print("if executed")

elif a%2==0:

print("1st elif executed")

elif a==5:

print("2nd elif executed")

else:

print("all conds fail")

======

Python Loops:

> for loops

> while loops

> do-while loops

for loops:

Syntax:

for var in sequence:

statements

Sequence =list,tupl,dic,set,string

Ex:p23.py

veh=["bike","car", "flight"]

for x in veh:

print(x)

Ex:p24.py

s="abcd"

for x in s:

print(x)

Note : else in for loop

Ex:p25.py

s="abcd"

for x in s:

print(x)

else:

print("for loop completed sucessfully")

Note : Builtin fun -=> range( )

it returns a valuees starting from 0 to range-1 with incm of 1 at a time

Ex :26.py

for x in range(6)

print(x) ============>displays starting from 0 to 5

Ex :27.py

for x in range(6,20)

print(x) =========> displays starting from 6 to 19

Ex :28.py

for x in range(3,30,4)

print(x) ==================>displays starting from 3 to 30 with incrementation of +4

===================

7/1/23

Note :

nested loops:

loop within a loop

Ex >> 29.py

vehi=[ "cycle" , "bike" , "car" ]

yr=[ "2001" , "2010" , "2020" ]

for x in vehi :

for y in yr :

print(x,y)

=====

>>while loop :

Syntax :

initialization

while cond:

statements

icrement/decerment

ex p30

i=10

while(i>0):

print(i)

i-=1

ex p31.py

i=10

while(i>0):

print(i)

i-=1

else:

print("while exec")

====

>>do while

this is wrong. there is no do while in python ;like c

this is renamed in py as

-while wiht break stmt

-while wiht exit stmt

ex 32.py

i=10

while(i>0):

print(i)

i-=1

if(i==5):

break

===========

>>break

==to terminate from current loop

== suntax : break

exx p33.py

vehi=[ "cycle" , "bike" , "car" ]

for x in vehi :

print(x)

if x=="bike":

break

====

>>continue:

to skip all stmts after continue but exec remains in the same loop

syn : continue

Ex : p34.py

==

pass

it is used to do nothing.

ex 35

for lopala pass

p36.py=palindrome

p37.py=display all pal bw 1 to 100

p38.py=prime

p39.py=to display all prime in 1 to 100

p40.py=to swap 2 numbers

=\==========/=/=/=/=/=/=/=/=/=/=//=/=/=/=/=//==/=

10/1/22

============

python functions:

it is defined as set of stmts or group of stmts or block of code which performs a specific task.

SYN:

def fn\_name(args) :

stmts

====p41====

def add(a,b):

return a+b

print(add(1,2))

===========

Advantages of fun:

>>reusabilit of code

>>educes the size of code

==========

classifications of fun:

>>user defined

>>built-in

>> user:

defined by users

def add(a,b):

return a+b

print(add(1,2))

>>built-in:

pre defined

ex:print, input, abs, range.....

x=input("enter a number")

20

print(x)

y=input()

30

print(y)

z=int(input())

2

print(z)

if anything othe than int like 9.22 then u r idiot :D

===================

types of fun's:

>>fn with no args and no return type

>>fn with no args and return type

>>fn with args and no return type

>>fn with args and return type

1) fn with no args and no return type:

def add():

a=int(input())

b=int(input())

print(a+b)

add()

=============

2) fn with no args and return type:

def add():

a=int(input())

b=int(input())

return (a+b)

print(add())

=============

3) fn with args and no return type:

def add(a,b):

print(a+b)

a=int(input())

b=int(input())

add(a,b)

===========

4) fn with args and return type:

def add(a,b):

return (a+b)

a=int(input())

b=int(input())

print(add(a,b))

==========

p43

special way:

def add(a:int,b:int)->int :

print(a+b)

a=int(input())

b=int(input())

add(a,b)

============

types of arg:

>>Default arguments:

====p44.py======

def add(a,b=20):

return a+b

print(add(1,2)) ===>3

print(add(1))=====>21

===p45=========

def add(a=10,b=20):

return a+b

print(add(1,2)) ===>3

print(add(1))=====>21

print(add())====>30

============

>>Keyword Arguments:

=======p6======

def student(name,r):

print(name,r)

(student("aaa",1))

(student("aaa",r=111))

(student(r=1,name="aaa"))

note :

they are known as named arguments..

=========

>>Required args:

no of args in fn call == no of args in fn

ex:

def student(name,r):

print(name,r)

(student("aaa",1))

===========

====p47=====

def add(a,b):

return a+b

add(10,20

all three...

============

>>4) arbitrary args:

it is called as variable length args

========p48==========

def dis(\*a):

for i in a:

print(i)

dis(1,2,3,4,5)

=================================

**Recursive functions:**

fun which calls itself==>recursive fun

**syntax:**

**def rec( ):**

**--------**

**-------**

**----------**

**Rec( )**