PySupPro

April 20, 2021

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

[ ]:
```
#Arithmetic operators
x = 4
y = 5
print(x + y)
print(x - y)
print(x * y)
print(x / y)

print(x % y)
print(x ** y)
print(x // y) #floor division
```

[ ]:
```
x=10
y=3
print(x/y)
print(x//y)
```

[ ]:
```
#Assignment operators
x = 5
x += 3
print(x) #5+3=8
x -= 3
print(x) #8-3=5
x *= 3
print(x) #5*3=15
x /= 3
print(x) #15/3=5.0
x %= 3
print(x) #5.0%3=2.0
```

[ ]:
```
x=10
x //= 3
print(x) #10//3=3
x **= 3
print(x) #3**3=27
```

[ ]:
```
print(15/3)
print(15//3)
print(2.0**3)
```

[ ]:
```
x=34
x &= 3
print(x)
x |= 3
print(x)
x ^= 3
print(x)
x >>= 3
print(x)
x <<=3
print(x)
```

[ ]:
```
#Comparison operators
x=9
y=5
print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

[ ]:
```
# Logical Operators
#AND-
#00 - 0
#01 - 0
#10 - 0
#11 - 1
x = 20
print(x < 6 and x > 25)
print(x < 6 and x < 25)
print(x > 6 and x > 25)
print(x > 6 and x < 25)
 #HOMEWORK- TRY or AND not
print(not(x > 6 and x < 25))
```

[ ]:
```
#Identity operators - is and not is
x = 4
y = 7
z = x
print(x is z)
print(x is not y)
print(x == y)

x = 4
y = 4
z = x
print(x is z)
print(x is y)
print(x == y)
```

[ ]:
```
# Membership operators(after lists) -> in/not in
```

[ ]:
```
#Bitwise operators
x=5
y=6
print(x&y) #and
print(x|y) #or
print(x^y) #xor
print(~x,~y) #not
print(x<<y) #left shift
print(x>>y) #right shift
```

[ ]:
```
x=10
y=7
print(x&y)
```

0.1 Boolean

[ ]:
```
print(10 > 9)
print(type(10 > 9))
```

[ ]:
```
print(bool("Hello"))
print(bool(-15))

print(bool("a"))
print(bool(1))

print(bool(""))
print(bool(0))

s="h"
y=0
```

```
print(bool(s))
print(bool(y))
```

[ ]:
```
print(bool(1))
print(bool(0))
```

The print has been removed from Python 2 as keyword and included as built-in function.

1. A variable name must start with a letter or the underscore character
2. A variable name cannot start with a number
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
4. Variable names are case-sensitive (age, Age and AGE are three different variables)
5. Variable names cannot contain keywords

[ ]:
```
import keyword
keyword.kwlist
```

myvar = “John”

my_var = “John”

_my_var = “John”

myVar = “John”

MYVAR = “John”

myvar2 = “John”

76trombones = “big parade”

more$ = 1000000

class = “Computer Science 101”

Markup language- writing standard, similar to Hypertext Markup Language(HTML) # heading 1 ## heading 2 ### heading 3 #### heading 4 ##### heading 5 ###### heading 6

This is a blockquote-

indented large text

> This is also a blockquote

$\sqrt{5}$

Bold Bold bold

Italic Italic Italic

a

b

c

a

b

c

0.1.1 Type Casting

[ ]:
```
x = 5
print(x)
x = "String"
print(x)
```

[ ]:
```
print(x,type(x))
x = int(3)
print(x,type(x))

y = int(3)
z = float(3)
print(x,y,z)
```

[ ]:
```
x = "A"
y = 'B'
print(x,y,type(x),type(y))
```

[ ]:
```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

[ ]:
```
a="A"
A ="B"
print(a,A)
```

[ ]:
```
a='A'
A ='ABC'
print(a,A)
```

Python floats are usually the equivalent of C doubles, and if the integer is too large, int() will cast it to a long int.

0.1.2 Input-Output

[ ]:
```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

[ ]:
```
num = input ("Enter number :")
print(num)
```

```
name = input("Enter name : ")
print(name)

print ("type of number", type(num))
print ("type of name", type(name))
```

[ ]:
```
num = int(input ("Enter number :"))
print(num)
name = input("Enter name : ")
print(name)

print ("type of number", type(num))
print ("type of name", type(name))
```

[ ]:
```
num1 = int(input ("Enter number :"))
print(num1)

num2 = float(input ("Enter number :"))
print(num2)
```

WORKING WITH STRINGS

[ ]:
```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

[ ]:
```
x= y= z = "Orange"
print(x)
print(y)
print(z)
```

[ ]:
```
b="hello world"
b
```

0.1.3 Square brackets can be used to access elements of the string.

0.1.4 Return a range of characters by using the slice syntax

string_name[start index:end index] - returns a part of the string.

[ ]:
```
b[0] #Get the character at position 0
```

[ ]:
```
print(b[0],b[4],b[-1])
```

[ ]:
```
b = "Hello, World!"
print(b[2:5])
```

[ ]:
```
b = "Hello, World!"
print(b[3:8])
```

[ ]:
```
b = "Hello, World!"
print(b[:5])
```

[ ]:
```
b = "Hello, World!"
print(b[3:])
```

[ ]:
```
b = "Hello, World!"
print(b[-1])
```

[ ]:
```
b = "Hello, World!"
print(b[::2])
```

[ ]:
```
b = "Hello, World!"
print(b[-5:-2])
```

[ ]:
```
print(len(b))
```

[ ]:
```
b = "Hello, World!"
print(b.lower())
```

[ ]:
```
b = "Hello, World!"
print(b.upper())
```

[ ]:
```
b = "Hello, World!"
print(b.isupper())
```

[ ]:
```
b="Value is positive"
print(b.index("s"))
print(b.index("z"))
```

[ ]:
```
a = "    Hello    ,      World!        "
print(a.strip()) # removes white spaces from beginning and ending of string
```

[ ]:
```
a = " Hello , World!"
print(a.split("o"))
```

[ ]:
```
b
b.upper()
```

[ ]:
```
b=b.upper()
b
```

[ ]:
```
print(a.replace("H", "J"))
```

[ ]:
```
print(len(b))
```

[ ]:
```
print(b[::-1])
```

[ ]:
```
#MULTILINE STRINGS
#ERROR
# a="Lorem ipsum dolor sit amet,
# consectetur adipiscing elit,
# sed do eiusmod tempor incididunt
# ut labore et dolore magna aliqua."

b='''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''

print(b,"\n")
    print("\n")

c="""Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""

print(c)
```

0.1.5 INDENTATION- whitespace at the beginning of a line to define scope in the code.

1 if-else

[ ]:
```
#VALUE IS POSITIVE/NEGATIVE/ZERO?
a = int(input())
if(a>0):
    print("POSITIVE")
else:
    print("NEGATIVE")
```

[ ]:
```
#VALUE IS POSITIVE/NEGATIVE/ZERO?
a = int(input())
if (a>0):
    print("POSITIVE")
elif (a<0): #else if
    print("NEGATIVE")
else:
    print("ZERO")
```

[ ]:
```
a = int(input())
if a > 0: print("POSITIVE")
elif (a<0): print("NEGATIVE")
else: print("ZERO")
```

1.0.1 Write a program to find the maximum of 3 numbers while asking user for input

[ ]:
```
a=int(input())
b=int(input())
c=int(input())

if(a>b) and (a>c):
    print("A")
    print(a,"is the greatest")

elif(b>a) and (b>c):
    print("B")
    print(b,"is the greatest")

elif(c>a) and (c>a):
    print("C")
    print(c,"is the greatest")

else:
    print("They are equal")
```

1.0.2 Write a program to find the if the input value is even or odd

A school has following rules for grading system:

a. Below 25 - F

b. 25 to 45 - E

c. 45 to 50 - D

d. 50 to 60 - C

e. 60 to 80 - B

f. Above 80 - A

Ask user to enter marks and print the corresponding grade.

[ ]:
```
marks = float(input())
if (marks<25) and (marks>=0):
    print ("F")
elif marks>=25 and marks<45:
    print ("E")
```

```
elif marks>=45 and marks<50:
    print ("D")
elif marks>=50 and marks<60:
    print ("C")
elif marks>=60 and marks<80:
    print ("B")
elif(marks>=90 and marks<=100):
    print ("A")
else:
    print("Enter valid number")
```

1.0.3 PRINT 1-5

1.0.4 While loop

[ ]:
```
i = 2
while i < 6:
    print(i)
    i += 1
```

[ ]:
```
i=5
while (i<501):
        print(i)
        i=i+2
```

1.1 for loop - used for iterating over a sequence

1.1.1 range() Function

[ ]:
```
for x in range(6):
    print(x)
```

[ ]:
```
for x in range(2, 6):
    print(x)
```

[ ]:
```
for x in range(2, 30, 3):
    print(x)
```

[ ]:
```
for x in "Something":
    print(x)
```

[ ]:
```
for x in range(0,6):
    print(x)
    if x == 3:
        break
```

[ ]:
```
for x in range(0,6):
    if x == 3:
        continue

    print(x)
```

[ ]:
```
for x in range(11):
    if (x==5):
        continue
    if(x==8):
        break
    print(x)
```

Python has two primitive loop commands:

while loops (execute a set of statements as long as a condition is true)

for loops

1.1.2 You have to store the names and marks of 5 students

There are 5 students: s1,s2,s3,s4,s5

Their marks are: m1,m2,m3,m4,m5

[ ]:
```
s1=input("Enter name of student 1 ")
m1=int(input("Enter marks of student 1 "))

s2=input("Enter name of student 2 ")
m2=int(input("Enter marks of student 2 "))

s3=input("Enter name of student 3 ")
m3=int(input("Enter marks of student 3 "))

s4=input("Enter name of student 4 ")
m4=int(input("Enter marks of student 4 "))

s5=input("Enter name of student 5 ")
m5=int(input("Enter marks of student 5 "))
```

Built-in data types - lists, sets, tuples, dictionaries

1.1.3 LISTS - used to store multiple items in a single variable

Syntax: List_name=[“item1”,“item2”..]

[ ]:
```
Names=["a","b","c","d","e"]
Marks=[45,34,32,64,93]

print(Names)
```

```
print(Marks)
```

[ ]:
```
Names=[]
Marks=[]

n=int(input("Enter number of students"))

for i in range(0,n):

    print("Enter names of student",i)
    a=input()

    print("Enter marks of student",i)
    b=int(input(""))

    Names.append(a)
    Marks.append(b)

print(Names)
print(Marks)
```

1. Lists can be indexed like strings
2. Can change/update their values
3. Append adds new values to the end of list
4. Lists allow duplicate values

[ ]:
```
#1
print(Names[0])
print(Names[0:2])

print(Marks[1:5])
print(Marks[-1])
```

[ ]:
```
#2
print(Names)
Names[0]="ABC"
print(Names)
```

[ ]:
```
#3
print(Names)
Names.append("XYZ")
print(Names)
```

[ ]:
```
#4
print(Names)
Names.append("XYZ")
Names[0]="XYZ"
```

```
print(Names)
```

[ ]:
```
print(len(Names))
print(len(Marks))
print(type(Names), type(Marks))
```

[ ]:
```
a = ["a", "b", "c"]
b = [1, 5, 7, 9, 3]
c = [True, True, False]
print(a,b,c)
```

1.2 Can list contain data of different data types? YES!

[ ]:
```
a = ["XYZ", 2, False, 40, "Nice"]
a
```

[ ]:
```
print(Names)
Names.remove("XYZ")
print(Names)
```

[ ]:
```
print(Marks)
Marks.remove(54)
print(Marks)
```

1.3 A couple of questions

http://pythontutor.com/

[ ]:
```
iteration = 0
count = 0
while iteration < 5:
    for letter in "hello, world":
        count += 1
    print("Iteration " + str(iteration) + "; count is: " + str(count))
    iteration += 1
```

[ ]:
```
for iteration in range(5):
    count = 0
    while True:
        for letter in "hello, world":
            count += 1
        print("Iteration " + str(iteration) + "; count is: " + str(count))
        break
```

[ ]:
```
a=10
print(type(str(a)))
print(type(a))
```

```
print("Printing",a)
print("Printing"+str(a))
print("Printing",str(a))
```

1. Define a boolean list and display it
2. Ask the user to input an index i and a val which is either 0 or 1
3. If the user inputs 0, at the index i, change the value to false
4. If the user inputs 1, at the index i, change the value to true
5. Display the list

[ ]:
```
B=[True,False,False,True]
print(B)

i=int(input("Enter index "))
val=int(input("Enter 0/1 "))

if (val==0):
    B[i]=False
elif (val==1):
    B[i]=True
else:
    print("Enter valid input ")
print(B)
```

1.4 TUPLES- Same as lists but they are unchangeable

[ ]:
```
Names=("a","b","c","d","e")
Marks=(45,34,32,64,93)

print(Names)
print(Marks)
```

[ ]:
```
a = ("xyz",)
print(type(a))

b = ("abc")
print(type(b))
```

[ ]:
```
print(Names[0])
print(Names[0:2])

print(Marks[1:5])
print(Marks[-1])
```

[ ]:
```
print(Names)
Names[0]="ABC"
```

```
print(Names)
```

[ ]:
```
print(len(Names))
print(len(Marks))
print(type(Names), type(Marks))
```

[ ]:
```
a = ("a", "b", "c")
b = (1, 5, 7, 9, 3)
c = (True, True, False)
d = ("XYZ", 2, False, 40, "Nice")

print(a,b,c,d)
```

Convert the tuple into a list to be able to change it

1.5 SETS

[ ]:
```
Set1 = {"a", "b", "c","c","c"}
print(Set1)
print(len(Set1))
```

[ ]:
```
Set2 = {"a", 4, True, 4.2, "B"}
print(type(Set2))
Set2
```

[ ]:
```
for i in Set2:
    print(i)
```

[ ]:
```
Set2.add(5.2)

print(Set2)
```

[ ]:
```
Set1.update(Set2)

print(Set1)
```

[ ]:
```
Set1
```

[ ]:
```
Set1.remove("a")
print(Set1)
```

[ ]:
```
Set1.remove("z")
print(Set1)
```

[ ]:
```
Set1.discard("z")
print(Set1)
```

DICTIONARIES: key:value pairs

[ ]:
```
Dict1 = {
  "a": 1,
  "b": 2,
  "c": 3
}
Dict1
```

[ ]:
```
print(Dict1["b"])
```

[ ]:
```
print(Dict1["a"])
```

[ ]:
```
Dict2 = {
  "a": 1,
  "b": 2,
  "c": 3,
    "d":3
}
Dict2
```

[ ]:
```
Dict3 = {
  "a": 1,
  "b": 2,
  "c": 3,
    "c":4
}
Dict3
```

[ ]:
```
print(len(Dict2))
print(len(Dict3))
```

1.6 Write a program to find the volume of a sphere

Store the values in a list

[ ]:
```
pi = 3.1415926535897931
r= int(input())
V= 4.0/3.0*pi* r**3
Vol=[]
Vol.append(V)
print(V,Vol)
```

1.7 Take input until 42 is inputed

[ ]:
```
i=0
while i!=42: i=int(input("Enter"))
```

[ ]:
```
s=int(input("Enter Input:"))
while(s!=32):
    s=int(input("Enter Input:"))
```

1.7.1 Count the number 4 in a given list

[ ]:
```
count = 0
nums=[2,4,53,6,466,9,4]
for num in nums:
    if num == 4:
        count = count + 1
print(count)
```

[ ]:
```
#42 91 68
#GAME - WORKS
x=int(input("Please think of a number between 0 and 100!"))
low=0
high=100
while(True):
    mid=(low+high)//2
    print('Is your secret number '+str(mid)+'?')
    w=input("Enter 'h' to indicate the guess is too high. Enter 'l' to indicate␣
 ↪the guess is too low. Enter 'c' to indicate I guessed correctly. ")
    if w =='h':
        high=mid
    elif w =='l':
        low=mid
    elif w=='c':
        print('Game over. Your secret number was: '+str(mid))
        break
    else:
        print("Sorry, I did not understand your input.")
```

1.7.2 FUNCTIONS

[ ]:
```
#def is a keyword
#is_even is the name of the function
#i is the parameter or argument
# Text in """""" is the DOCSTRING or the specification of the function

def is_even(i):

```

```
    '''
    Input: i is a positive int
    Returns True if is even otherwise False
    '''

    print("You wanna know??? ", i) #BODY OF FUNCTION
    return (i%2==0)

print(is_even(56))
is_even(53)

#56 AND 53 ARE ARGUMENTS
#I IS THE FUNCTION PARAMETER

#Parameter - name of the variables defined in the function signature and that␣
 ↪we use in the function's body. When we work inside a function, we work with␣
 ↪parameters.

#Argument - value or values that we call the function with, so it is the value␣
 ↪being passed into a function call.
```

[ ]:
```
def sum(a,b):
    s=a+b
    return s

a=int(input())
b=int(input())
print(sum(a,b))
```

1. Unique elements from a list
2. Function to find square of a number

[ ]:
```
def Unique_L(l):
    L2 = []
    for i in l:
        if i not in L2:
            L2.append(i)
    return L2

n=int(input("Enter number of elements"))
L1=[]
for i in range(0,n):
    a=input()
    L1.append(a)

Unique_L(L1)
```

[ ]:
```
def Unique_L(List1):
    s1=set(L1)
    l=list(s1)
    return l

n=int(input("Enter number of elements"))
L1=[]
for i in range(0,n):
    a=input()
    L1.append(a)

Unique_L(L1)
```

[ ]:
```
L1=[3,2,5,6,6,4]
s1=set(L1)
l=list(s1)
print(s1,L1,l)
```

[ ]:
```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for i in adj:
    for j in fruits:
        print(i, j)
```

The outer loop to print the number of rows. The inner loops to print the number of columns.

[ ]:
```
n = int(input("Enter the number of rows"))
for i in range(0, n):
        for j in range(0, i + 1):
            print("* ", end="")

        print()
```

[ ]:
```
def is_even(a,i):

    print("You wanna know??? ", a) #BODY OF FUNCTION
    return (i%2==0), 2 , i<55
    print("jfs") #will not be executed after return statement

print(is_even("john",56))
is_even("san",53)
```

[ ]:
```
def f_a():
    print('This function does not have any parameter list')
def f_b(y):
    print('This function does have one parameter')
```

```
    return y
print(f_a())
print(5+f_b(2))
```

We can access a variable defined outside but can't modify its value

[ ]:
```
def f(y):
    x=1 #since we defined x here, this is like a new object
    x+=1
    print(x)
x=5
f(x)
print(x)
```

[ ]:
```
def g(y):
    print(x,end=' ')
    print(x+1,end=' ')
x=5
g(x)
print(x)
```

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

[ ]:
```
def FNAME(a, b):
    print(a + " " + b)

FNAME("Hey")
```

[ ]:
```
def FNAME(a, b):
    print(a + " " + b)

FNAME("Hey","trh","fghd")
```

1.7.3 Arbitrary Arguments or *args This way the function will receive a tuple of arguments

[ ]:
```
def FNAME(*a):
    print(a[1])

FNAME("Hey","trh","fghd")
```

key = value syntax so that order of the arguments does not matter.

[ ]:
```
def FNAME(a, b, c):
    print(c)
```

```
FNAME(a = "Hey", b = "gfe", c = "sgv")
```

[ ]:
```
def FNAME(Food = "a"):
  print("I like " + Food)

FNAME("b")
FNAME("c")
FNAME()
FNAME("d")
```

Passing a List as an Argument

[ ]:
```
def FNAME(a):
    for i in a:
        print(i)

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

FNAME(a)
```

[ ]:
```
def FNAME(x):
    return 5 * x

print(FNAME(32))
print(FNAME(24))
print(FNAME(3))
```

Empty function

[ ]:
```
def FNAME():
    pass
```

[ ]:
```
s="This is a sentence"
l=s.split(" ")
l,type(l)
```

[ ]:
```
data="This is a sentence"
s=data.lower()
l=s.split(" ")
l
```