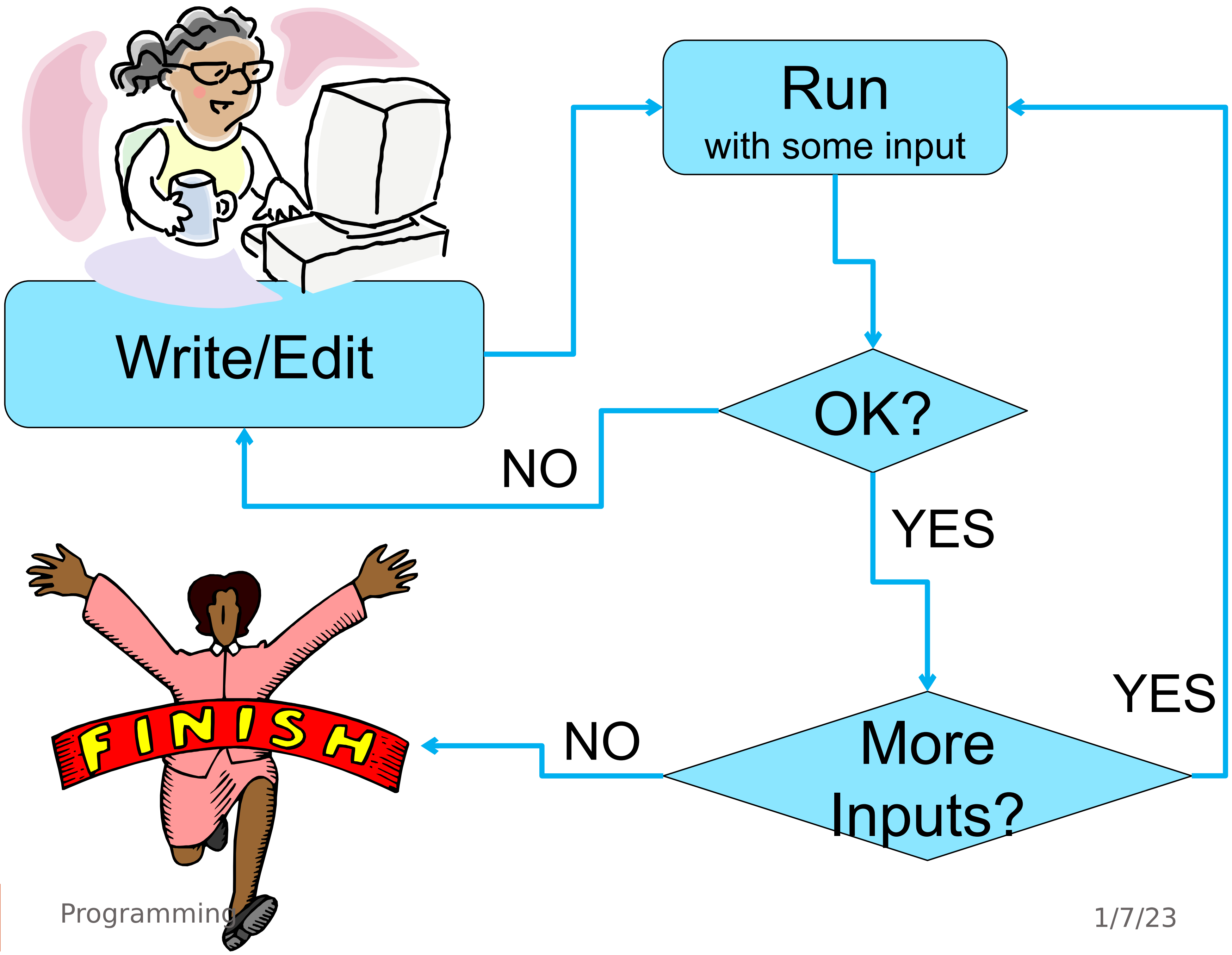
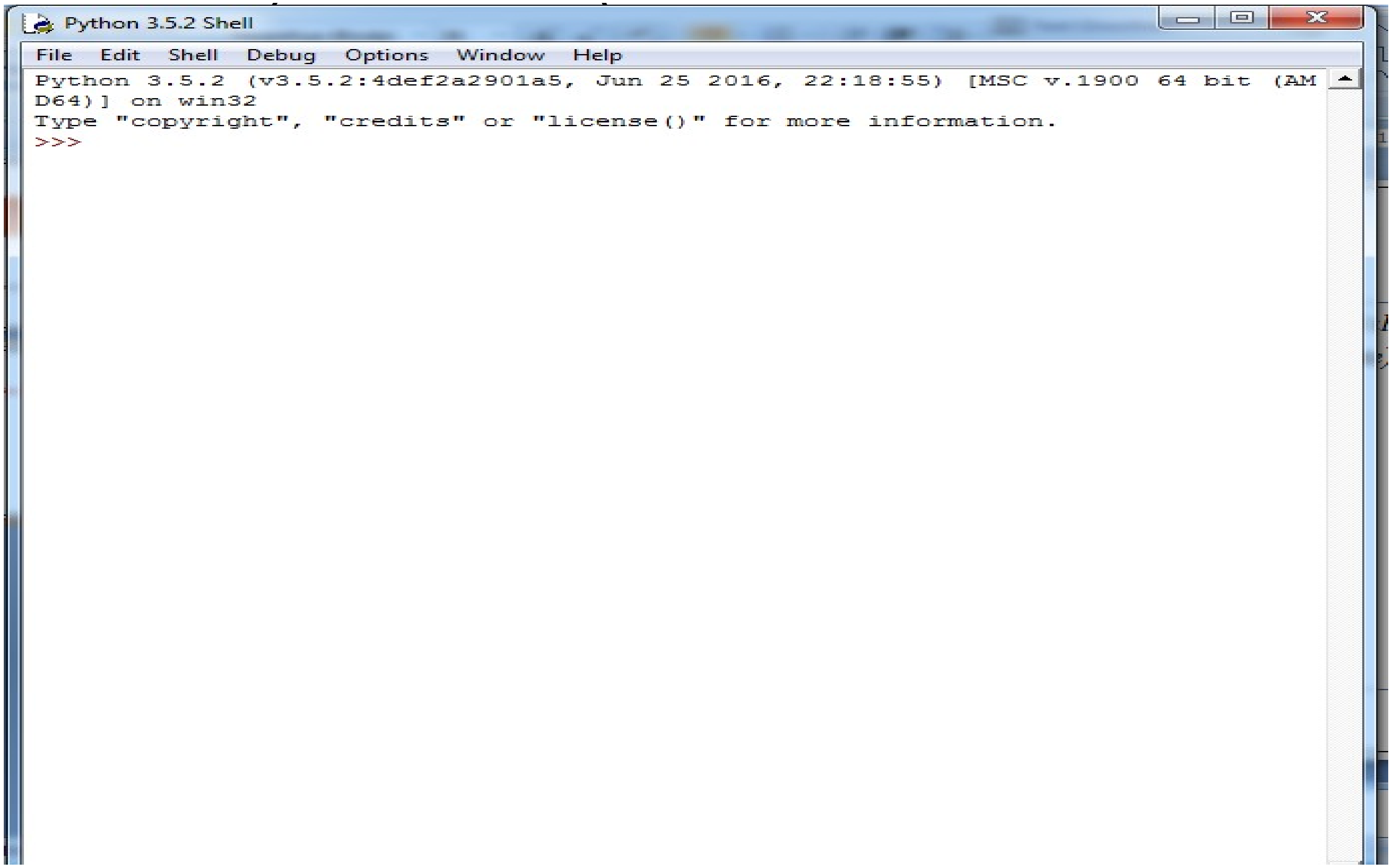
Python basics

Akshata S. Bhayyar
Assistant Professor, Dept. of CSE, MSRIT


Write/Edit
Run
with some input
OK?
NO
YES
More Inputs?
YES
NO
FINISH

Basics

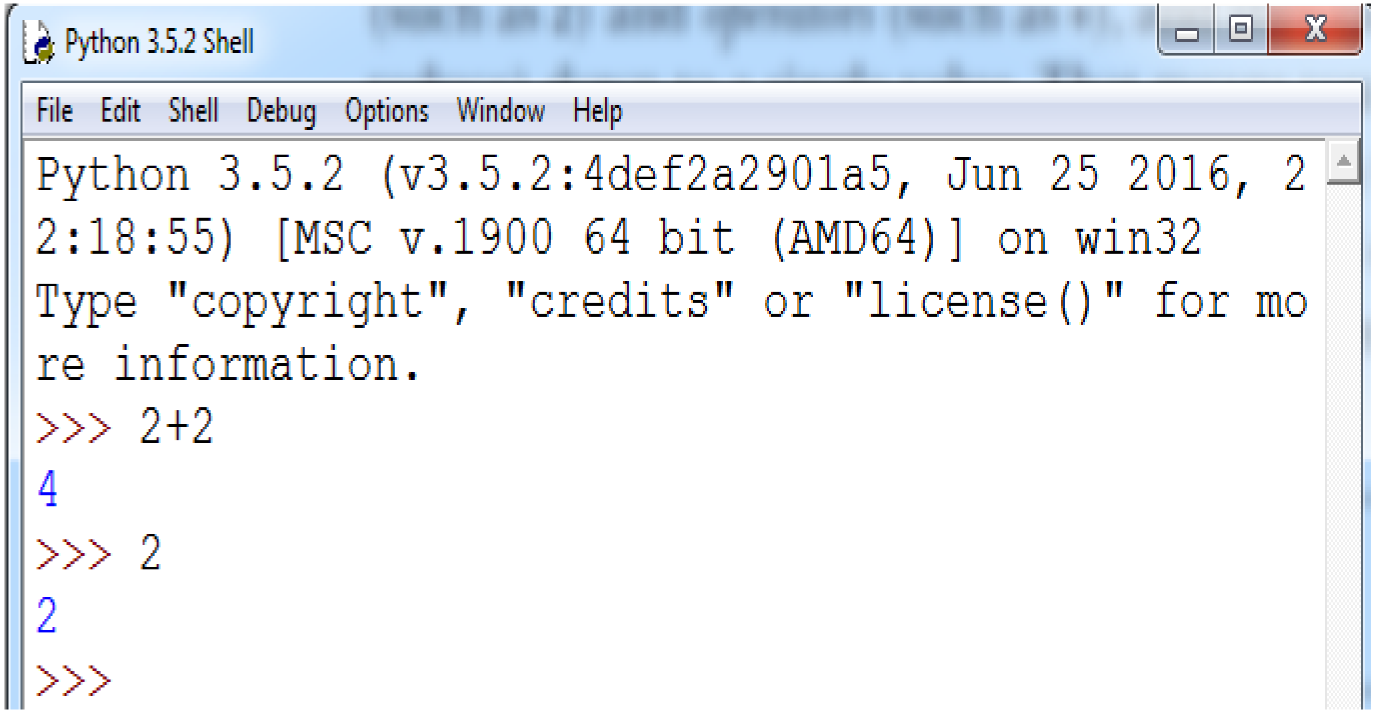
- Type into the *interactive shell, also called the REPL (Read-Evaluate-Print Loop), which* lets



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Basics

- Python instructions one at a time and instantly shows you the results.
- Expression
  - Values
  - operators



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 2
2:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for mo
re information.
>>> 2+2
4
>>> 2
2
>>>
```

Basics

- Error

- A *crash just means* the program stopped running unexpectedly.

Operators in python expressions

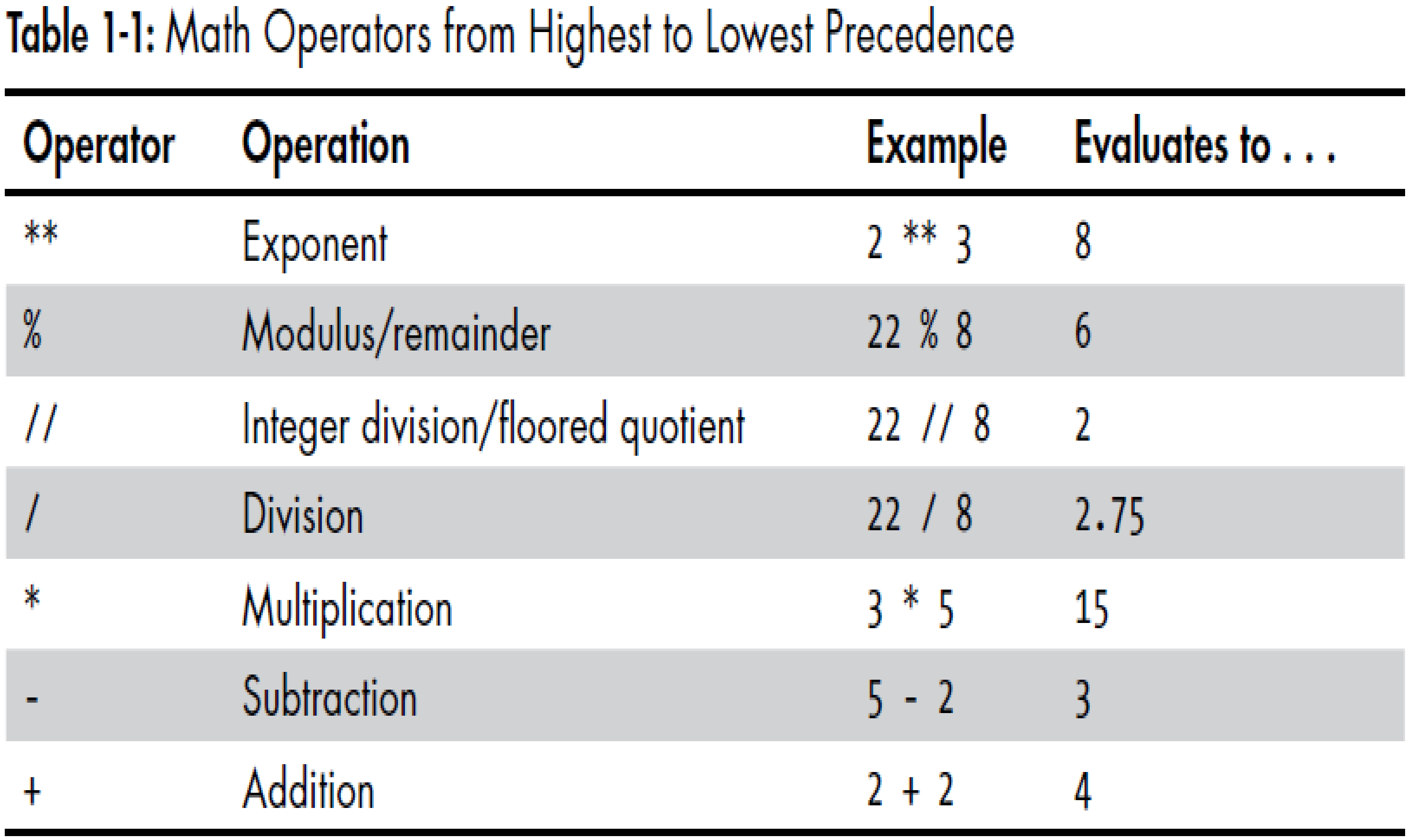
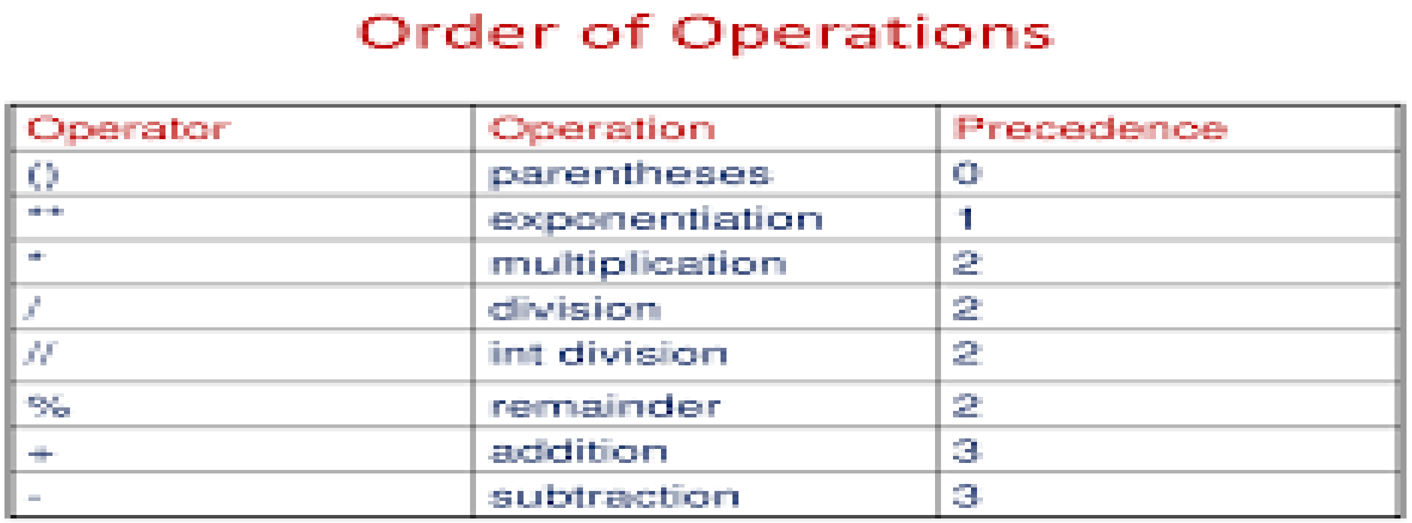


Table 1-1: Math Operators from Highest to Lowest Precedence

Operator	Operation	Example	Evaluates to . . .
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

Order of operations

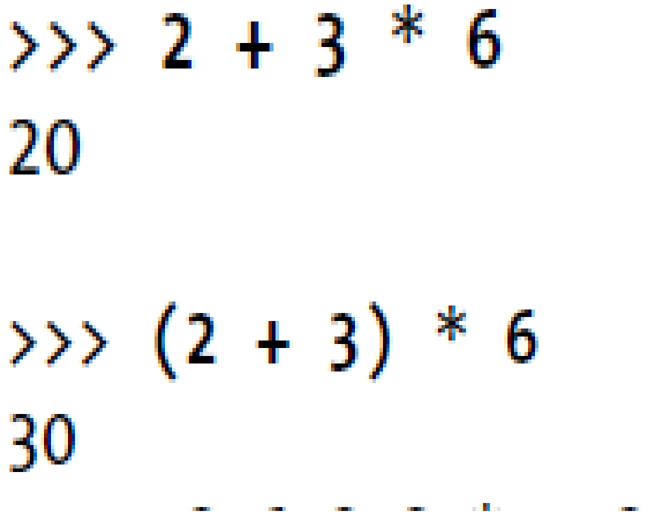


Order of Operations

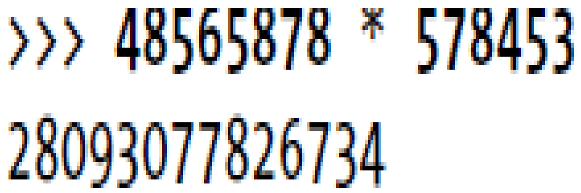
Operator	Operation	Precedence
()	parentheses	0
**	exponentiation	1
*	multiplication	2
/	division	2
//	int division	2
%	remainder	2
+	addition	3
-	subtraction	3

- Whitespace in between the operators and values doesn’t matter for Python (except for the indentation at the beginning of the line), but a single space is convention.

Order of operations

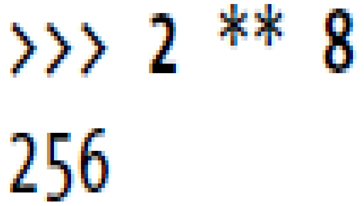


```
>>> 2 + 3 * 6
20

>>> (2 + 3) * 6
30
```

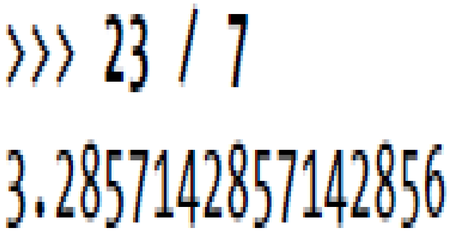


```
>>> 48565878 * 578453
28093077826734
```

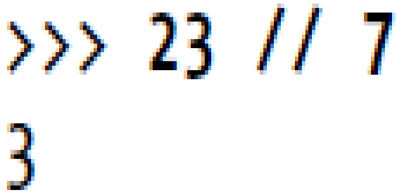
Order of operations



```
>>> 2 ** 8
256
```



```
>>> 23 / 7
3.2857142857142856
```

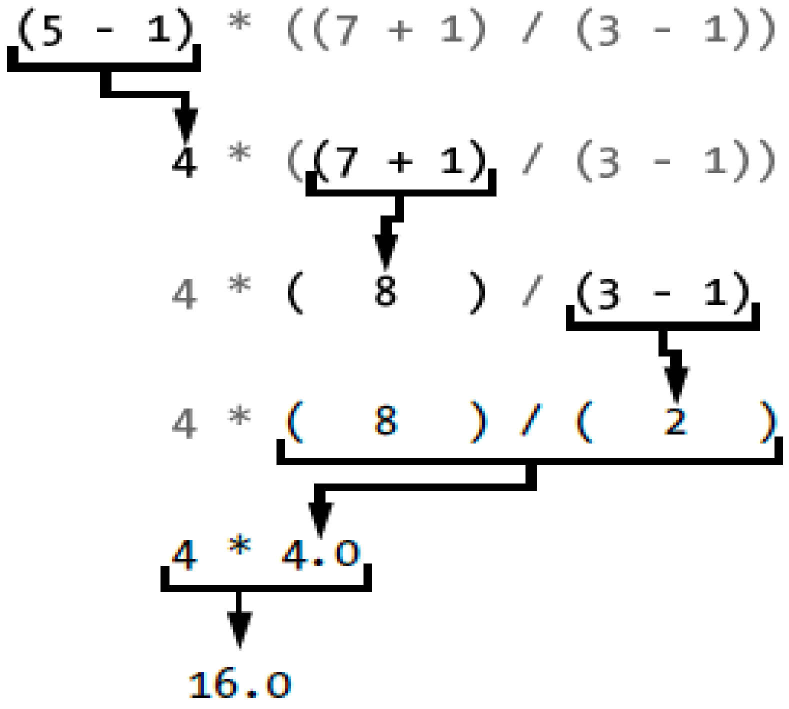


```
>>> 23 // 7
3
```

Order of operations

```
>>> 23 % 7
2

>>> 2      +          2
4


>>> (5 - 1) * ((7 + 1) / (3 - 1))

        16.0
```

Example

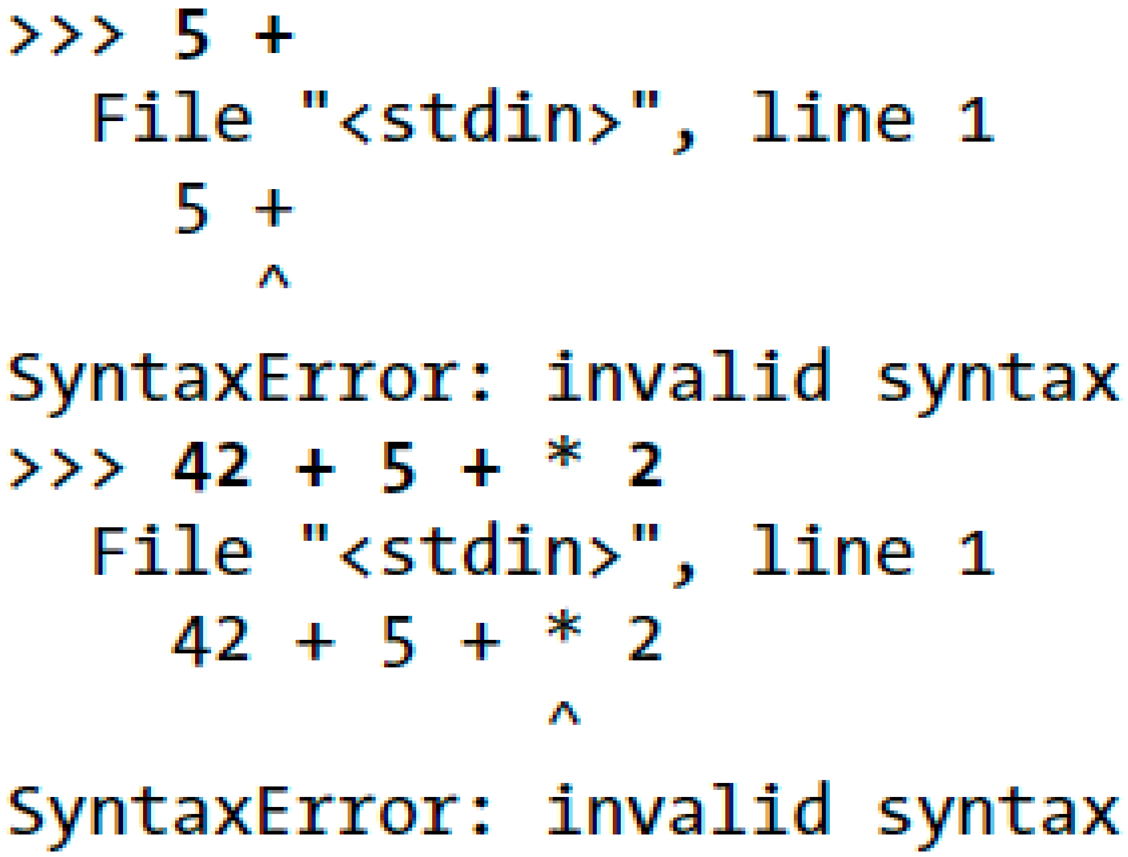


More Examples

A=10 B=-5 C=2 D=5

1. (A+B)*(C-D)
2. A//C%3
3. D*(B+A)
4. C+(B-C)/D

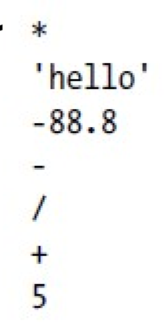
Syntax Error



```
>>> 5 +
  File "<stdin>", line 1
    5 +
      ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
  File "<stdin>", line 1
    42 + 5 + * 2
             ^
SyntaxError: invalid syntax
```

Questions

- Which of the following are operators, and which are values?



```
*
'hello'
-88.8
-
/
+
5
```

- What is an expression made up of?

The Integer, Floating-Point, and String Data Types

- A ***data type*** *is a category for values,* and every value belongs to exactly one data type

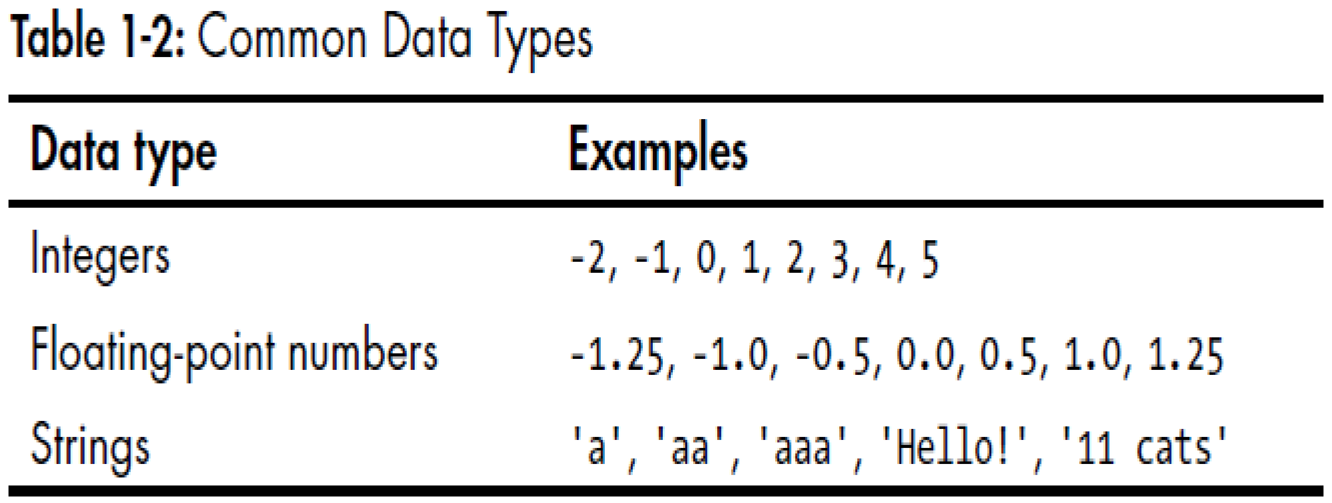
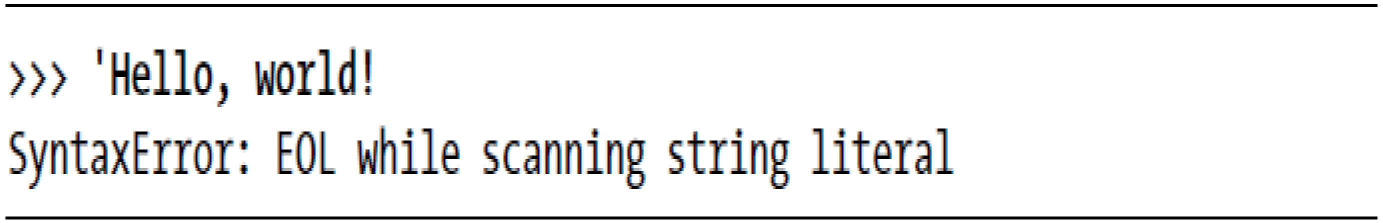


Table 1-2: Common Data Types

Data type	Examples
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 cats'

The Integer, Floating-Point, and String Data Types

- Python programs can also have text values called *strings*
- Always surround your string in single quote **(‘)**
- **Eg** 'Hello‘
- You can even have a string with no characters in it, **‘ '**, called *a blank string or an empty string.*



```
>>> 'Hello, world!
SyntaxError: EOL while scanning string literal
```

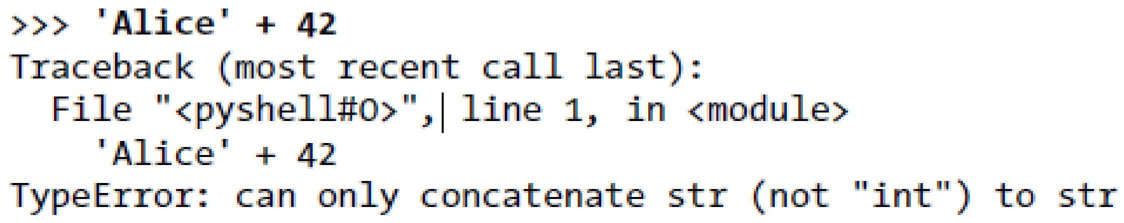
String Concatenation and Replication

- The meaning of an operator may change based on the data types of the values next to it.
- **Eg 2+3,   3.2 + 5.6**
- However, when + is used on two string values, it joins the strings as the *string concatenation operator*

>>> **'Alice' + 'Bob'**

'AliceBob‘

- However, if you try to use the + operator on a string and an integer value, Python will not know how to handle this, and it will display an **error message.**



```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    'Alice' + 42
TypeError: can only concatenate str (not "int") to str
```

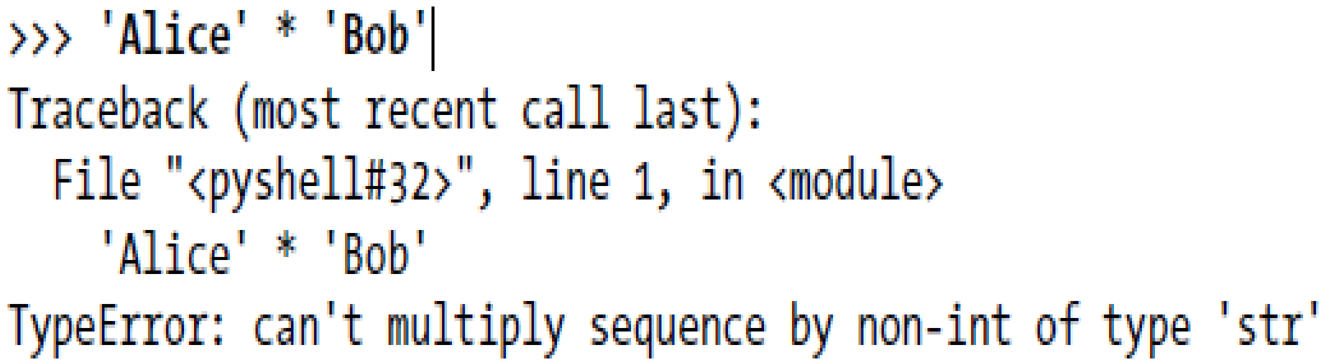
String Concatenation and Replication

- The * operator multiplies two integer or floating-point values.
- But when the * operator is used on one string value and one integer value, it becomes the *string replication operator.*

>>> 'Alice' * 5

'AliceAliceAliceAliceAlice‘

>>>



```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
```

Storing Values in Variables

What is variable ?

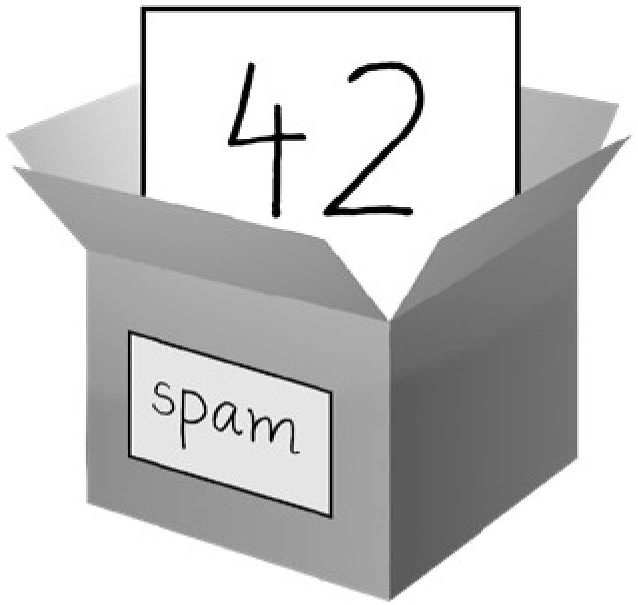
A **variable** is a quantity that may be changed according to the mathematical problem.

Eg : x+1

- A *variable is like a box in the computer's memory where you can store a* single value.

Assignment Statements

- An assignment statement consists of a variable name, an equal sign (called the *assignment operator), and the value to be stored.*
- *Eg* spam = 42



Assignment Statements

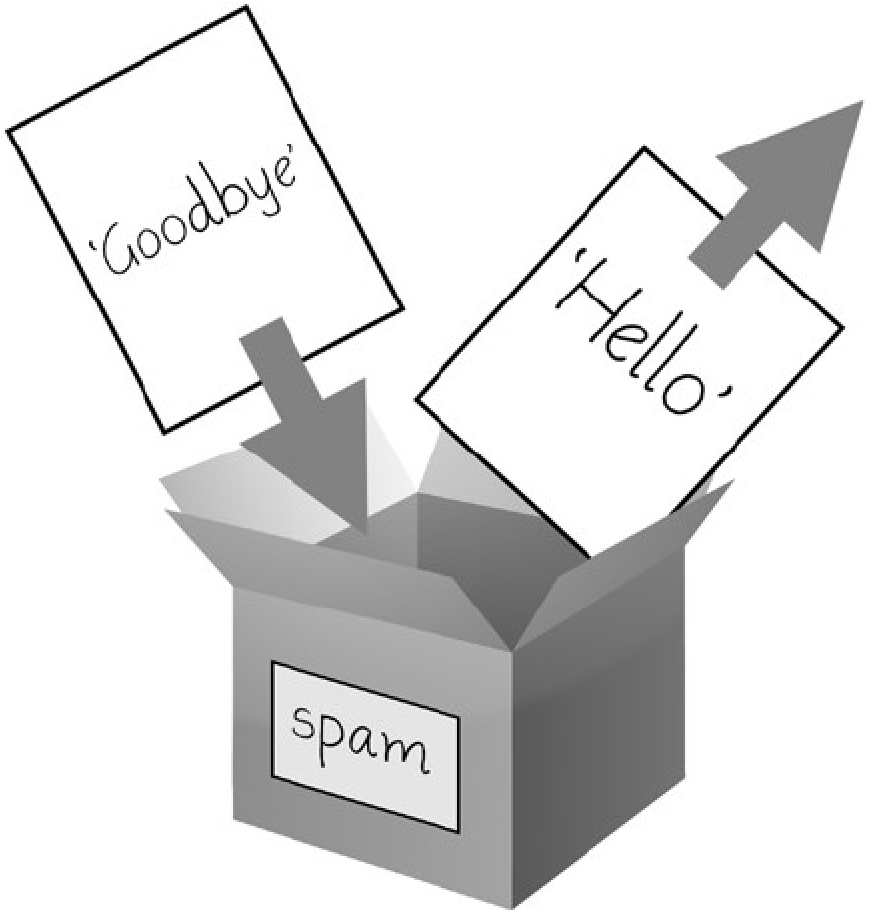
```
❶ >>> spam = 40
   >>> spam
   40
   >>> eggs = 2
```

```
❷ >>> spam + eggs
   42
   >>> spam + eggs + spam
   82
```

```
❸ >>> spam = spam + 2
   >>> spam
   42
```

variable *initialization*

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```



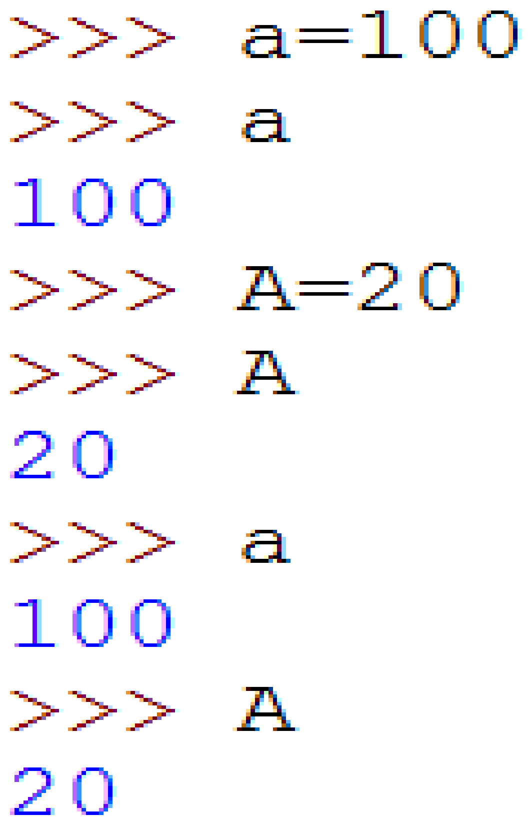
Variable Names

- A good variable name describes the data it contains
- Example?

- Naming restrictions

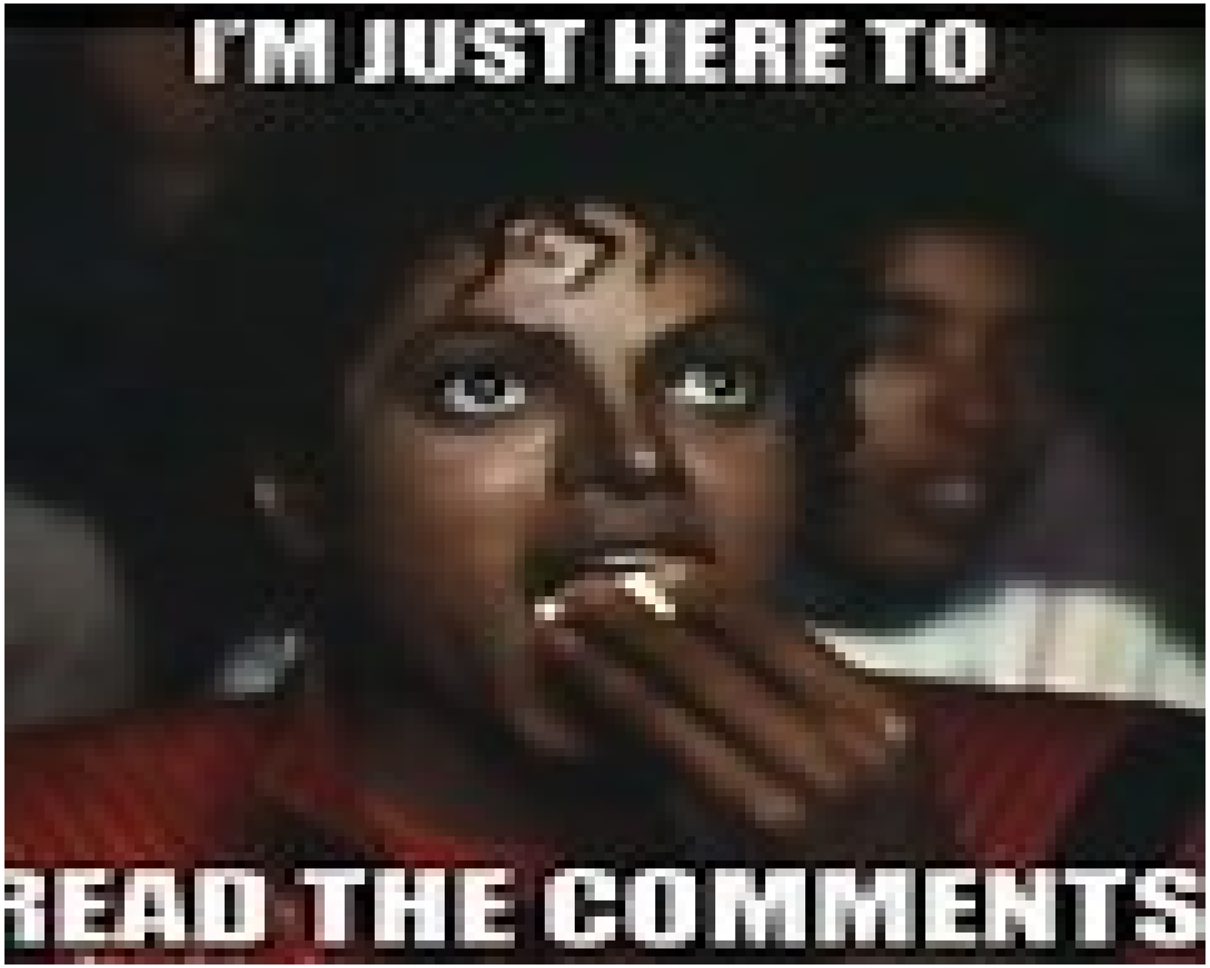
1. It can be only one word with no spaces.
2. It can use only letters, numbers, and the underscore (_) character.
3. It can’t begin with a number.
4. Variable names are case-sensitive, meaning that spam, SPAM, Spam, and sPaM are four different variables.
5. Python convention to start your variables with a lowercase letter.

Variable Names



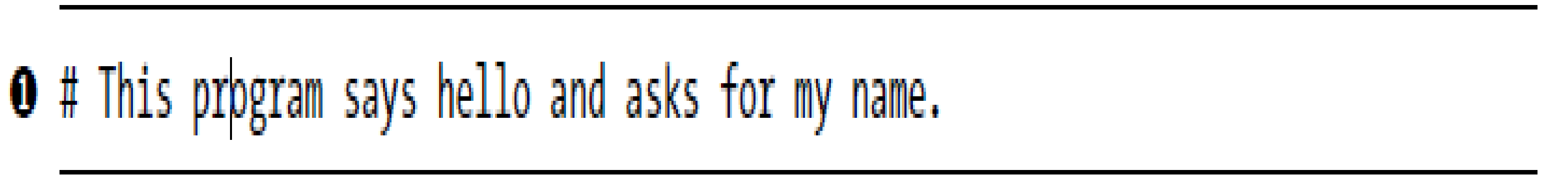
```
>>> a=100
>>> a
100
>>> A=20
>>> A
20
>>> a
100
>>> A
20
```

Comments



Comments

- The following line is called a *comment.*



```
❶ # This program says hello and asks for my name.
```

- Python ignores comments, and you can use them to write notes or remind yourself what the code is trying to do.

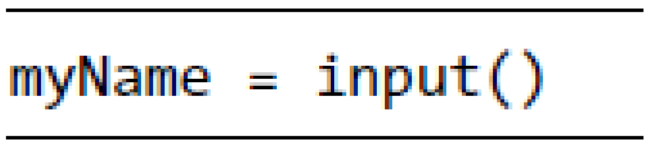
The print() Function

- The print() function displays the string value inside its parentheses on the screen.
- A value that is passed to a function call is an *argument.*
- *Notice that* the quotes are not printed to the screen.
- They just mark where the string begins and ends; they are not part of the string value.

- print( )?

The input() Function

- The input() function waits for the user to type some text on the keyboard and press enter.



```
myName = input()
```

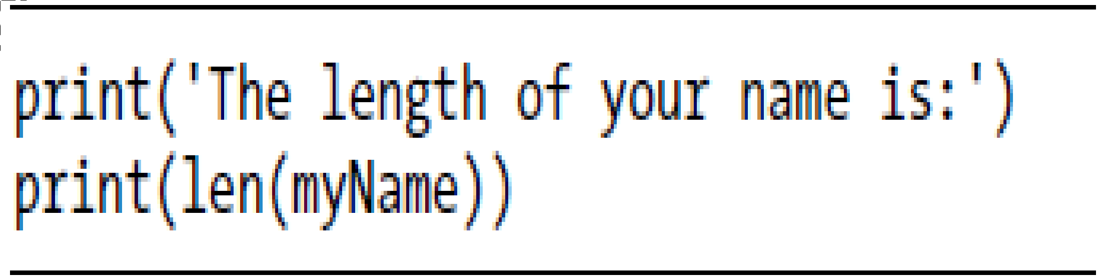
- Whatever you enter as input, the input function converts it into a string.
- If you enter an integer value still input() function convert it into a string.

Printing the User’s Name

```
print('It is good to meet you, ' + myName)
```

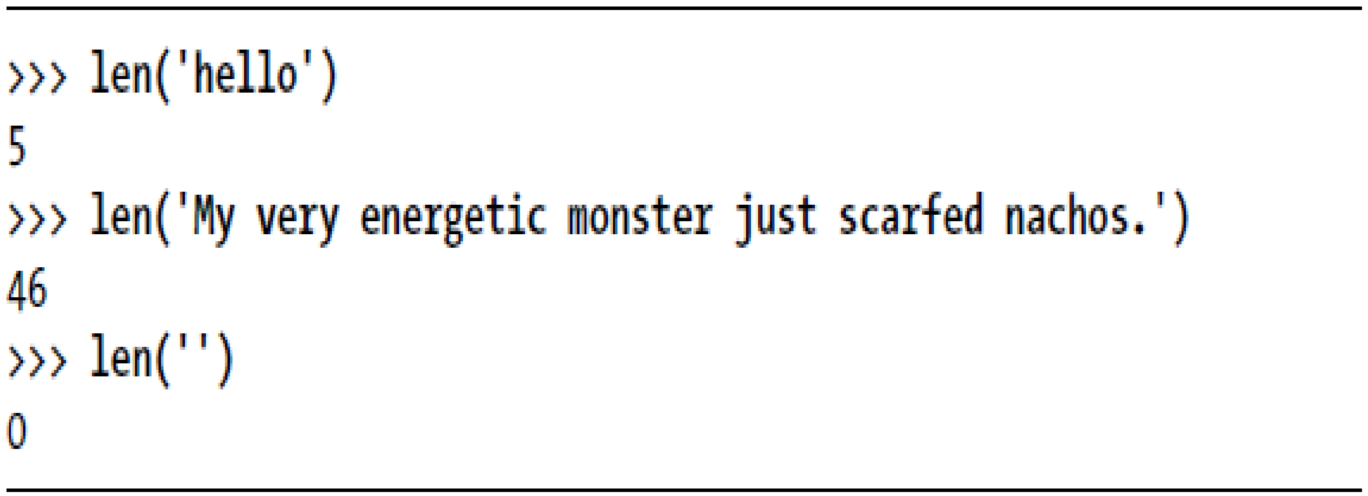
The len() Function

- You can pass the len() function a string value (or a variable containing a string), and the function evaluates to the integer value of the number of characters in that string



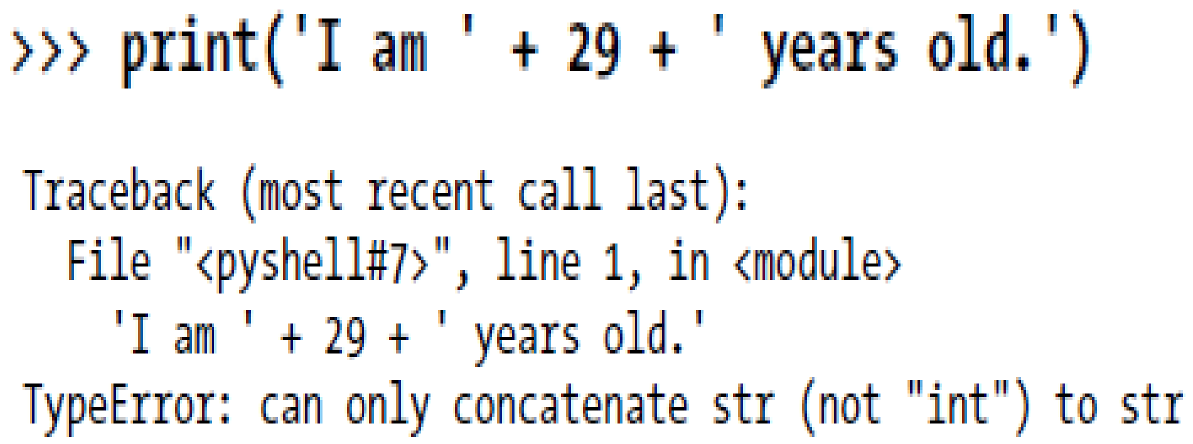
```
print('The length of your name is:')
print(len(myName))
```

The len() Function



```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
>>> len('')
0
```

The len() Function



```
>>> print('I am ' + 29 + ' years old.')

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'I am ' + 29 + ' years old.'
TypeError: can only concatenate str (not "int") to str
```

❑Python gives an error because the + operator can only be used to add two integers together or concatenate two strings.

Questions

- Name three data types.
- Which of the following is a variable, and which is a string?

```
spam
'spam'
```

- What does the variable bacon contain after the following code ru

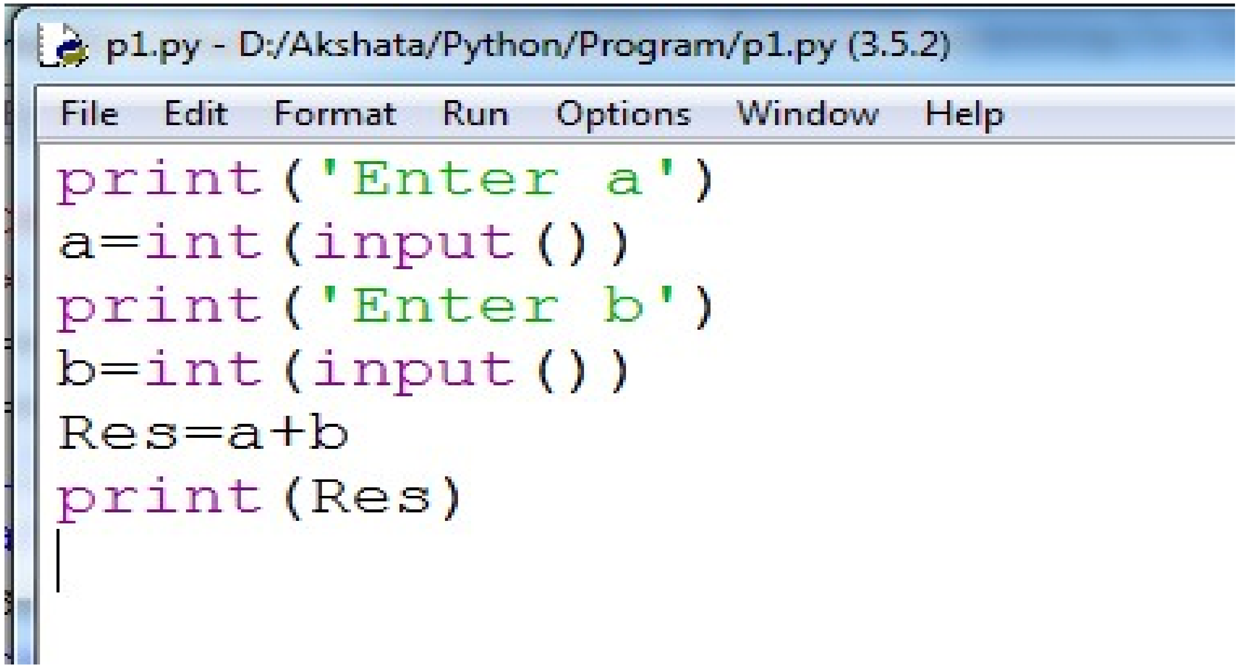
```
bacon = 20
bacon + 1
```

- What should the following two expressions evaluate to?

```
'spam' + 'spamspam'
'spam' * 3
```

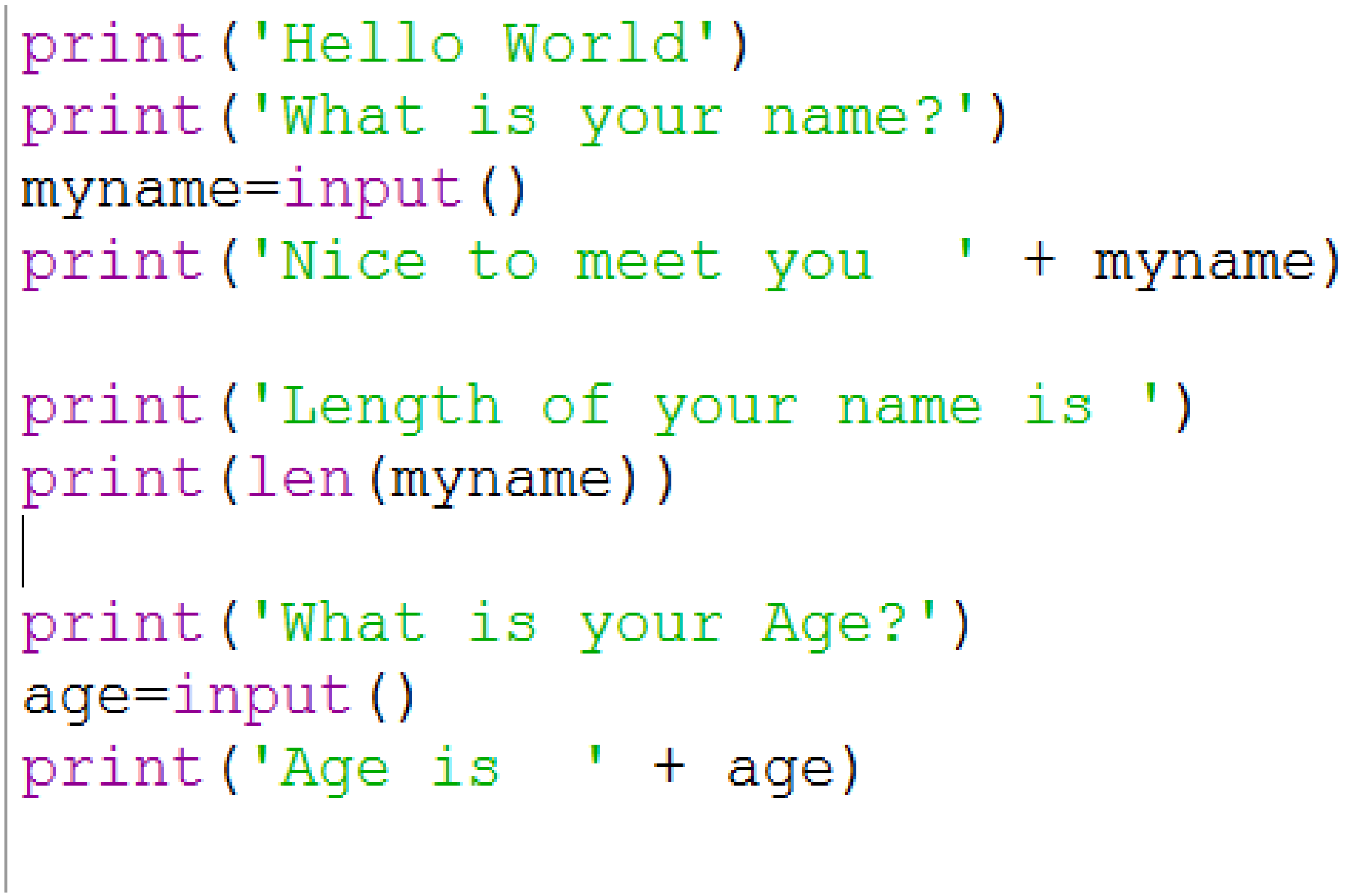
- Why is eggs a valid variable name while 100 is invalid?

First Program


p1.py - D:/Akshata/Python/Program/p1.py (3.5.2)
File Edit Format Run Options Window Help

```
print('Enter a')
a=int(input())
print('Enter b')
b=int(input())
Res=a+b
print(Res)
```

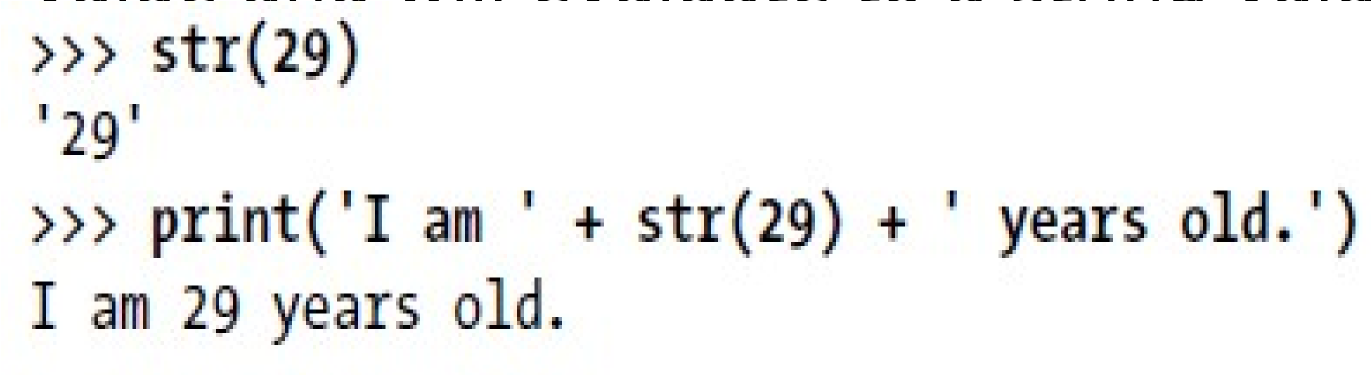
Second Program



```python
print('Hello World')
print('What is your name?')
myname=input()
print('Nice to meet you  ' + myname)

print('Length of your name is ')
print(len(myname))

print('What is your Age?')
age=input()
print('Age is  ' + age)
```

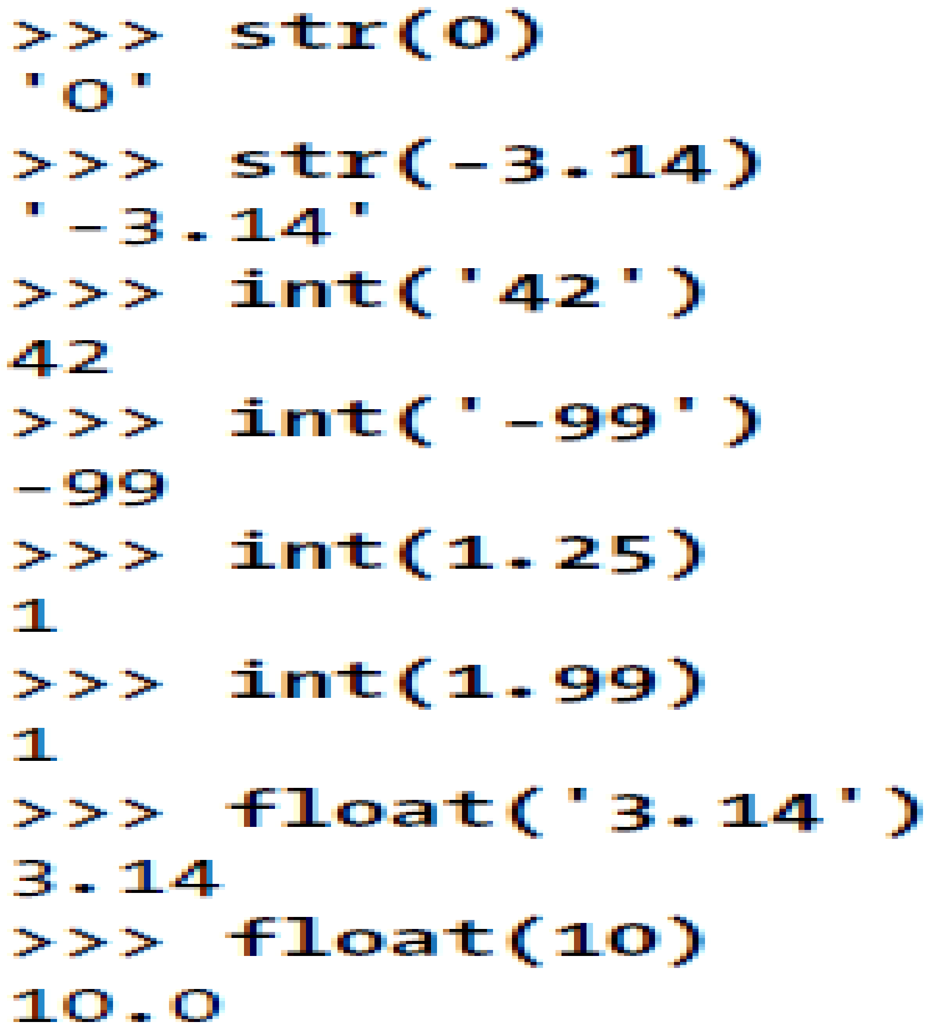
The str(), int(), and float() Functions

- If you want to concatenate an integer such as 29 with a string to pass to print(), you’ll need to get the value '29', which is the string form of 29.
- The str() function can be passed an integer value and will evaluate to a string value



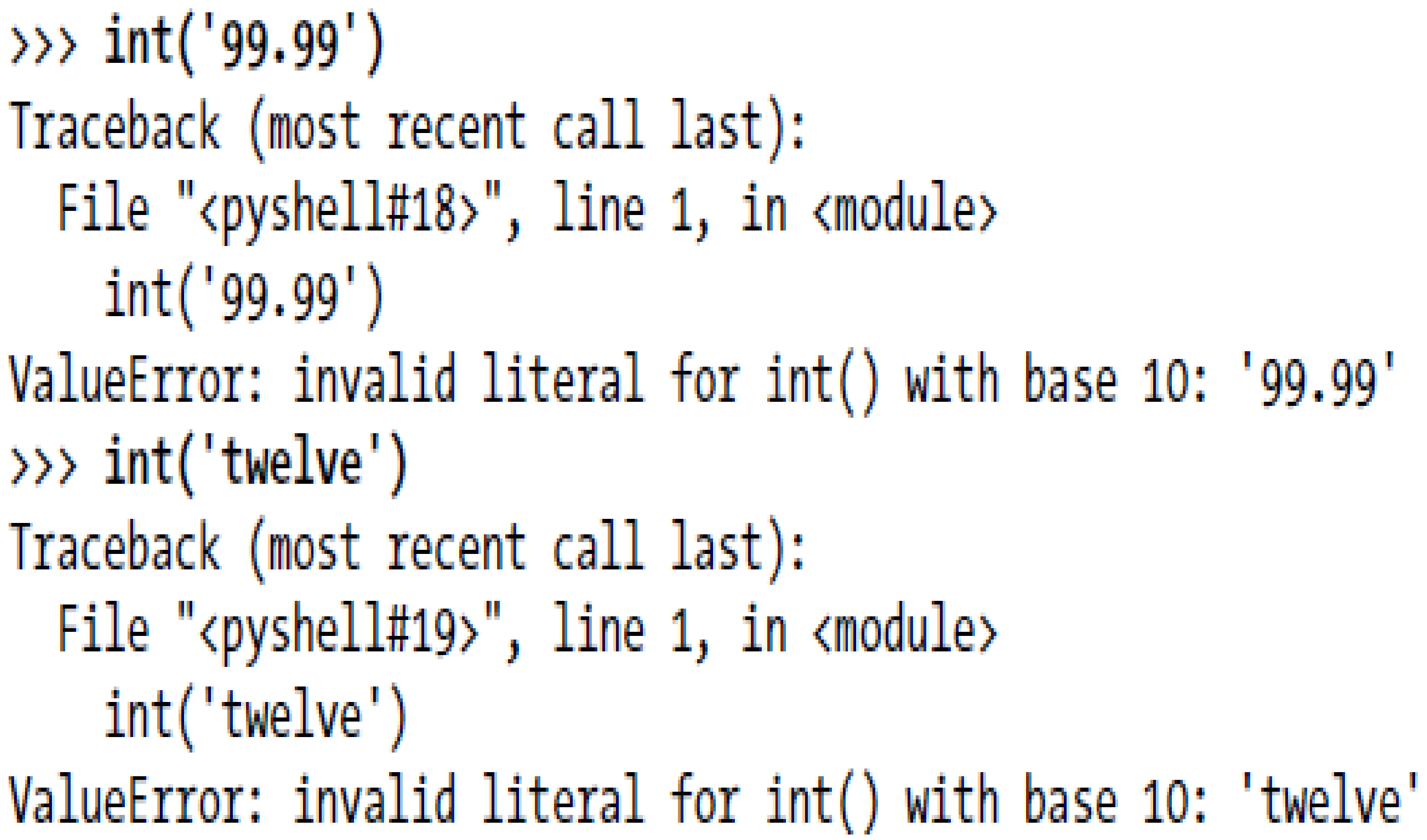
```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

The str(), int(), and float() Functions



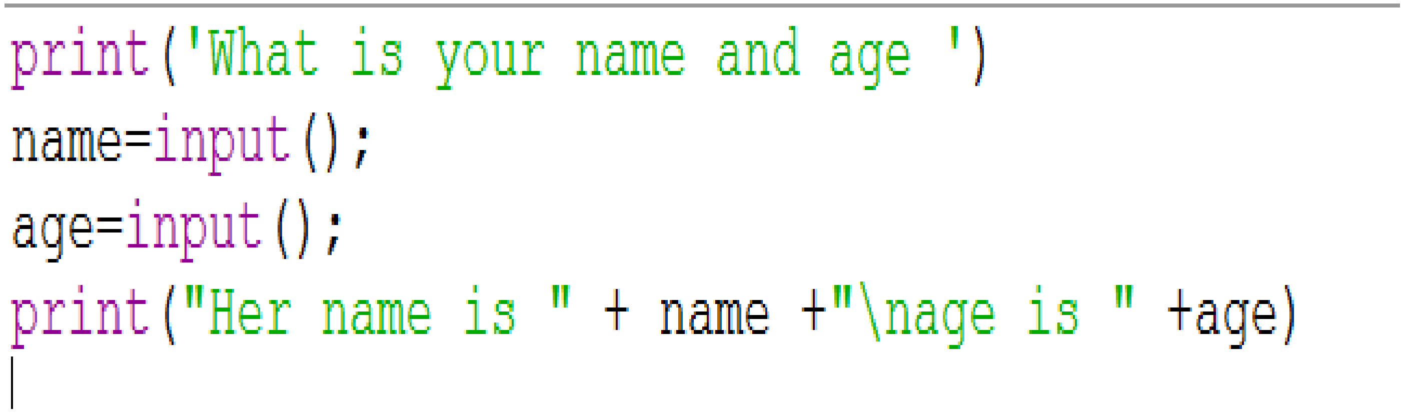
```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

The str(), int(), and float() Functions

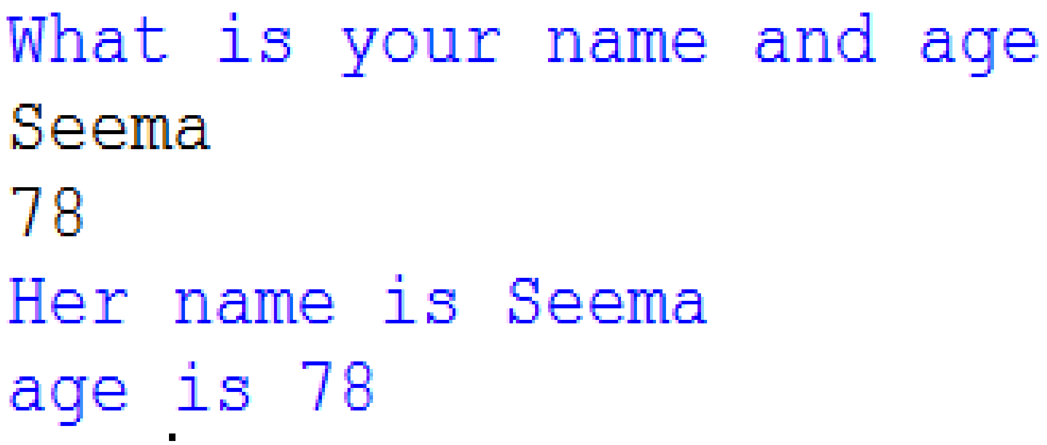


```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

Questions 1



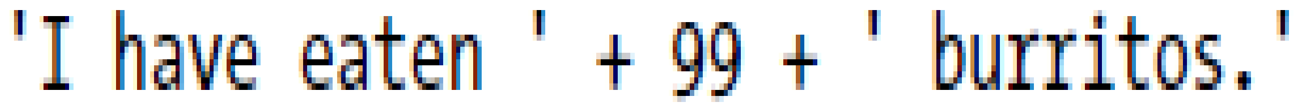
```
print('What is your name and age ')
name=input();
age=input();
print("Her name is " + name +"\nage is " +age)
```



```
What is your name and age
Seema
78
Her name is Seema
age is 78
```

Question

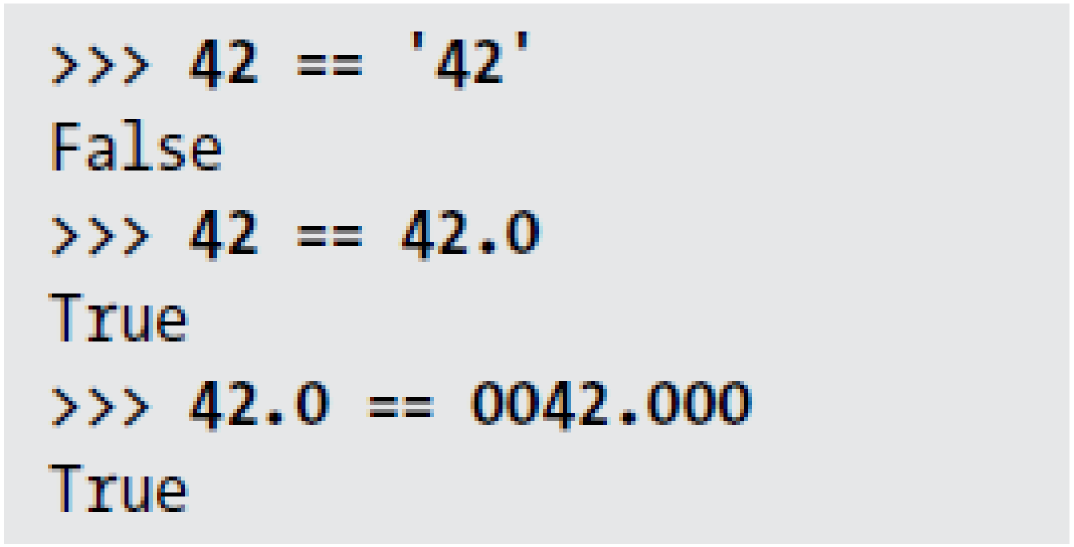
- Why does this expression cause an error? How can you fix it?



```
'I have eaten ' + 99 + ' burritos.'
```

TEXT AND NUMBER EQUIVALENCE

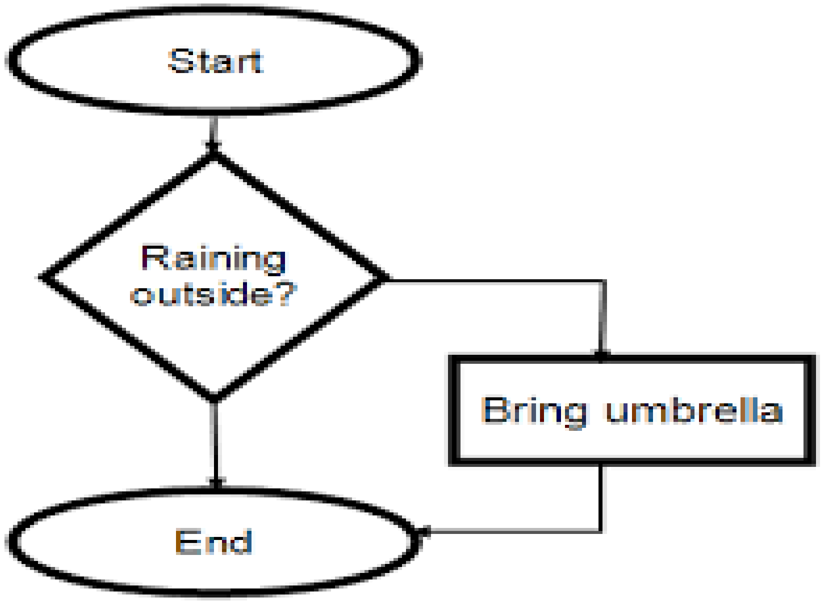
- ==   Equivalence operator



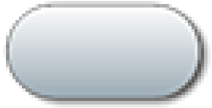
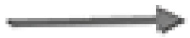
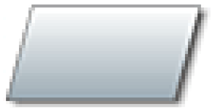
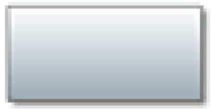
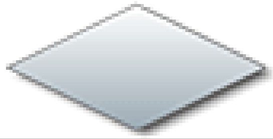
```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

FLOW CONTROL

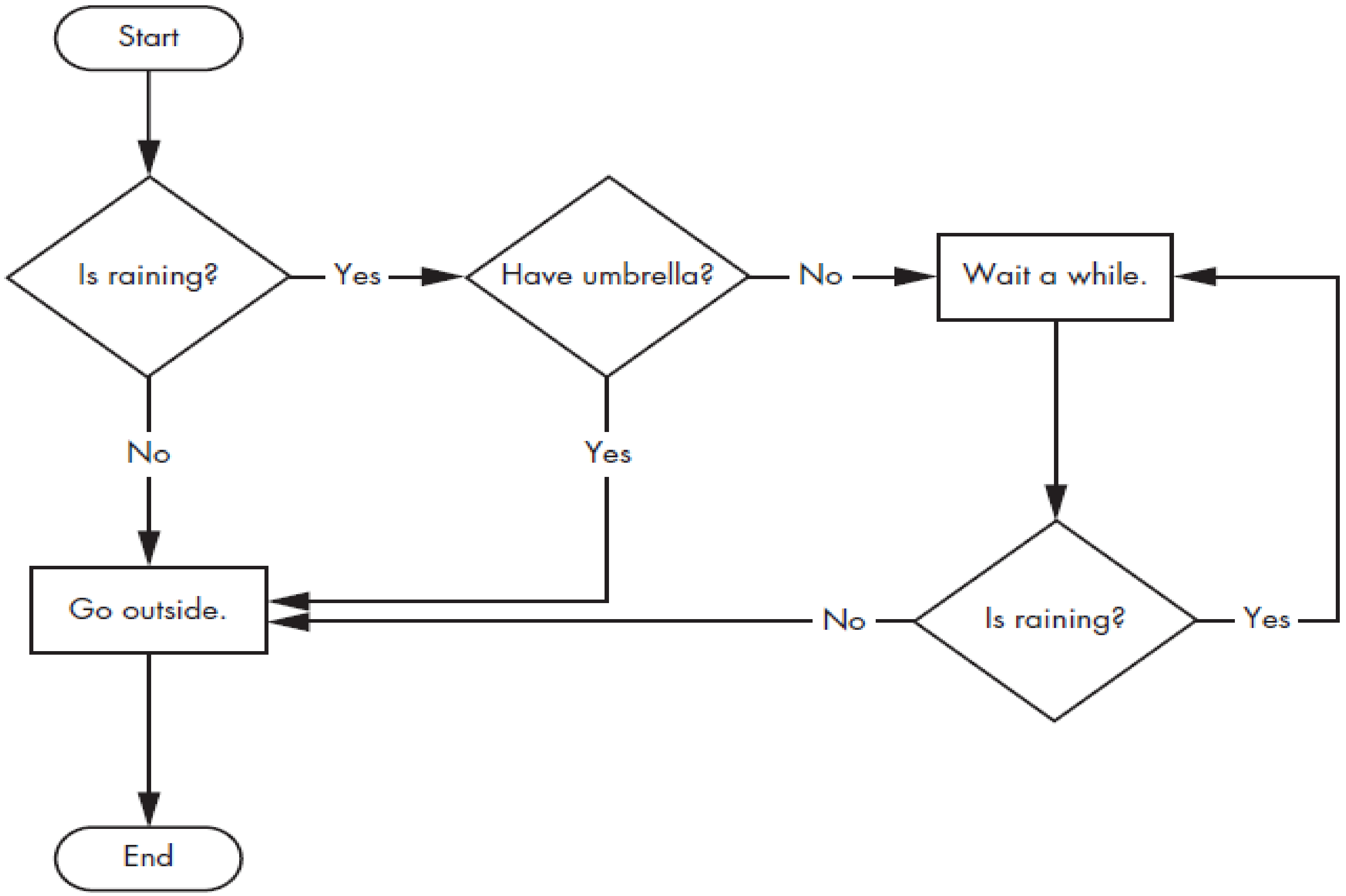
- Program is just a series of instructions.
- *Flow control statements can decide which Python instructions to execute under* which conditions.



FLOW CHART SYMBOL

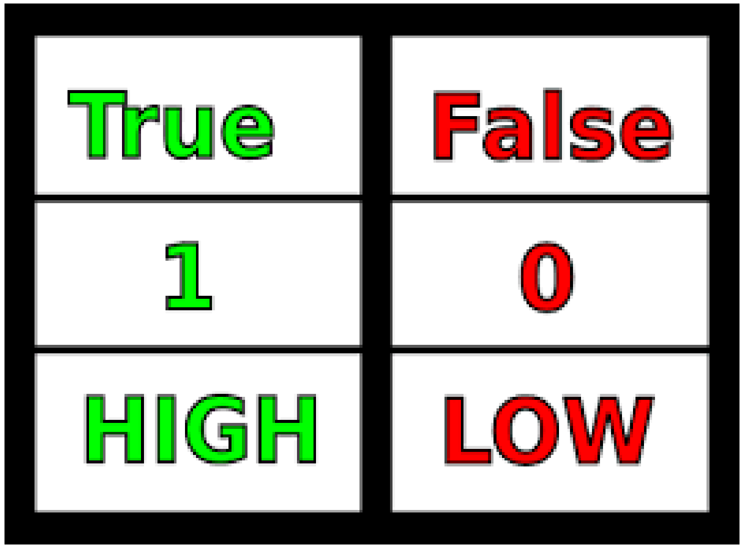
Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Arrows	A line is a connector that shows relationships between the representative shapes.
	Input/Output	A parallelogram represents input or ouptut.
	Process	A rectangle represents a process.
	Decision	A diamond indicates a decision.

FLOW CONTROL



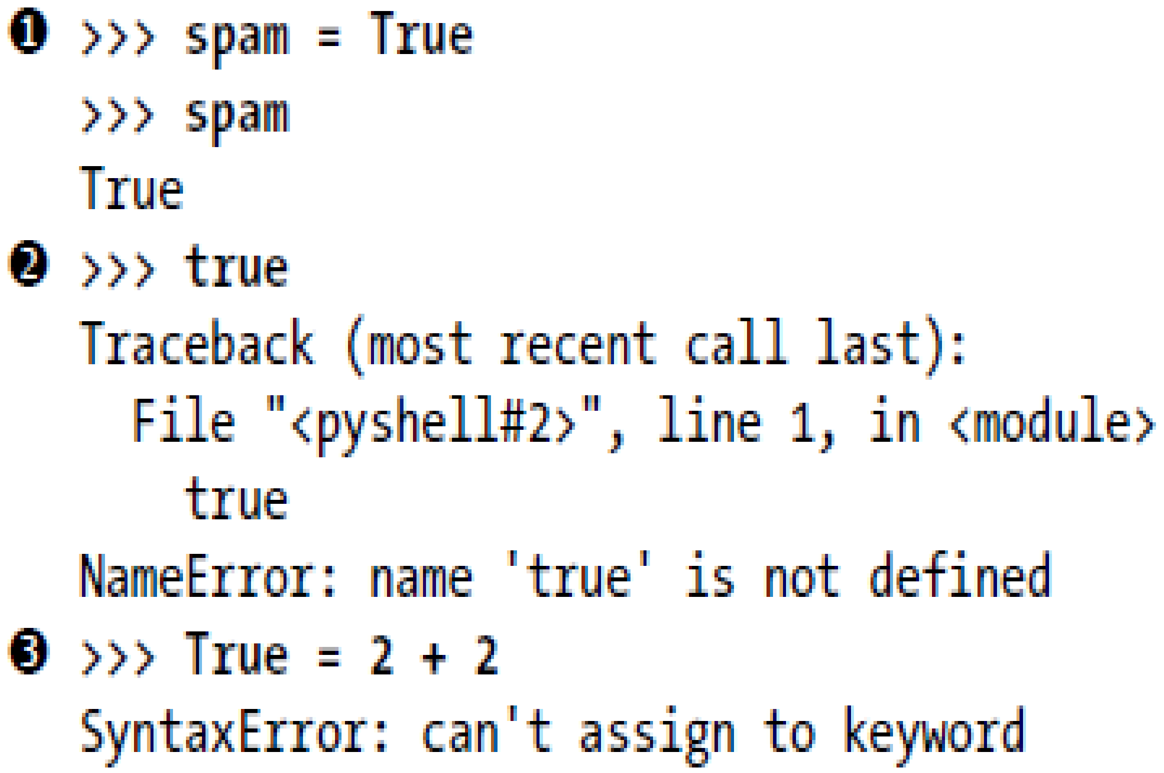
Boolean Values

They always start with a capital *T or F, with the rest of the word in lowercase.*



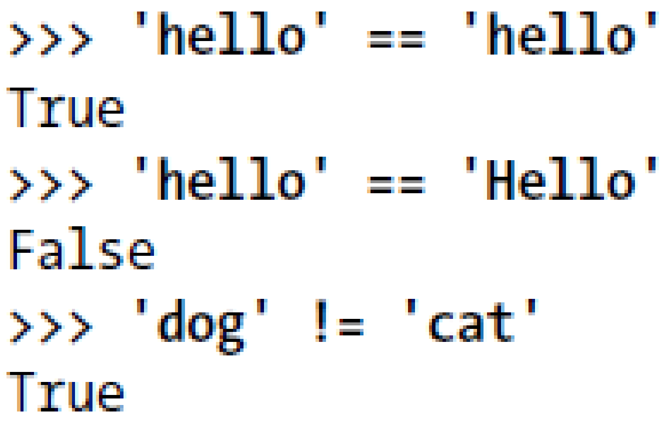
True	False
1	0
HIGH	LOW

Boolean Values



```
❶ >>> spam = True
  >>> spam
  True
❷ >>> true
  Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
      true
  NameError: name 'true' is not defined
❸ >>> True = 2 + 2
  SyntaxError: can't assign to keyword
```

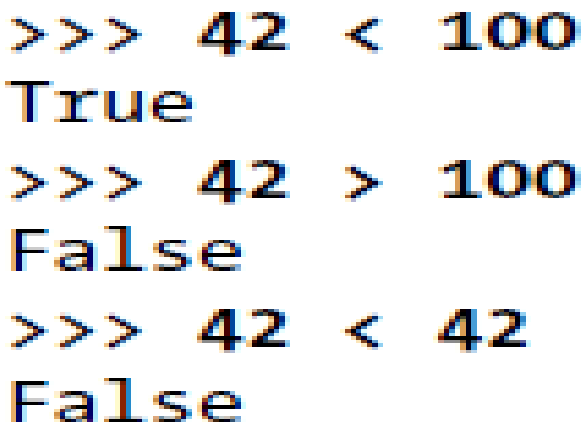
Boolean Values



```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
```

Boolean Values

The <, >, <=, and >= operators, on the other hand, work properly only with integer and floating-point values.



```
>>> 42 < 100
True
>>> 42 > 100
False
>>> 42 < 42
False
```

Question

Q THE DIFFERENCE BETWEEN THE == AND = OPERATORS ?

Boolean Operators

The three Boolean operators

- ✓**and**
- ✓or
- ✓not

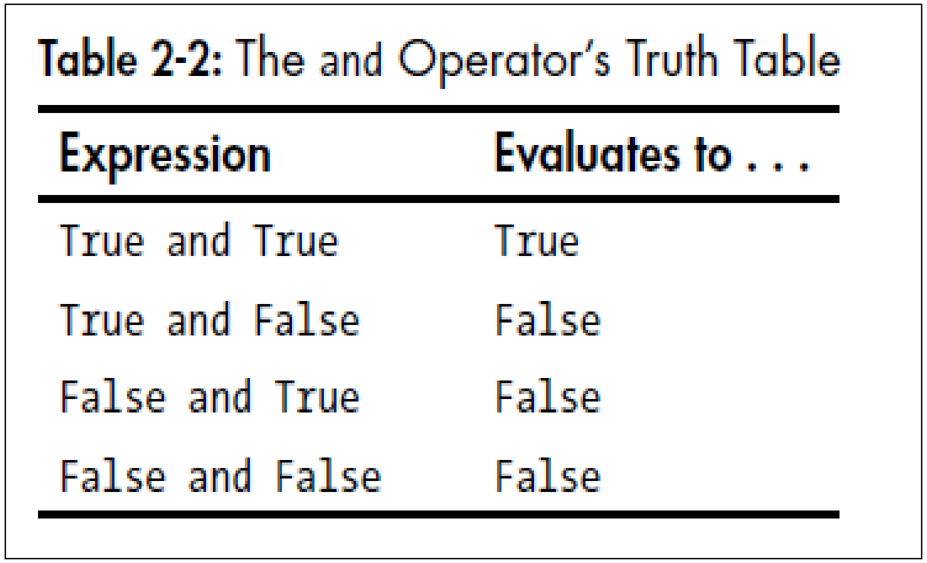


Table 2-2: The and Operator's Truth Table

Expression	Evaluates to . . .
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

```
>>> True and True
True
>>> True and False
False
```

Boolean Operators

The three Boolean operators

- ✓and
- ✓**or**
- ✓not

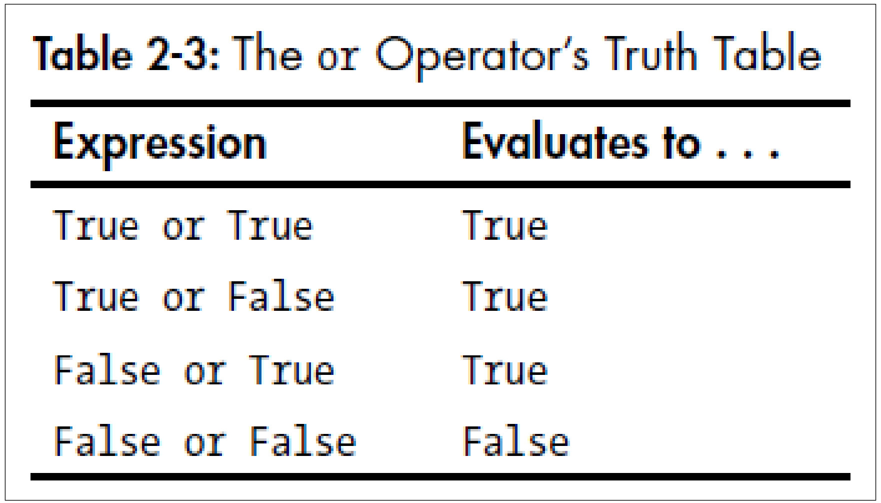
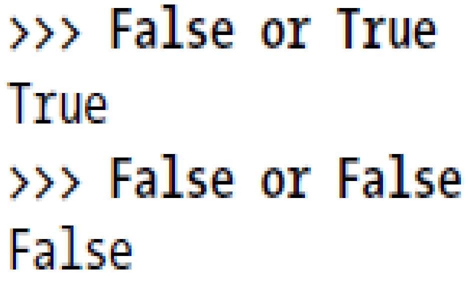


Table 2-3: The or Operator's Truth Table

Expression	Evaluates to . . .
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`



```
>>> False or True
True
>>> False or False
False
```

Boolean Operators

The three Boolean operators

- ✓and
- ✓or
- ✓**not**
  - ✓only one Boolean value- ***unary operator***

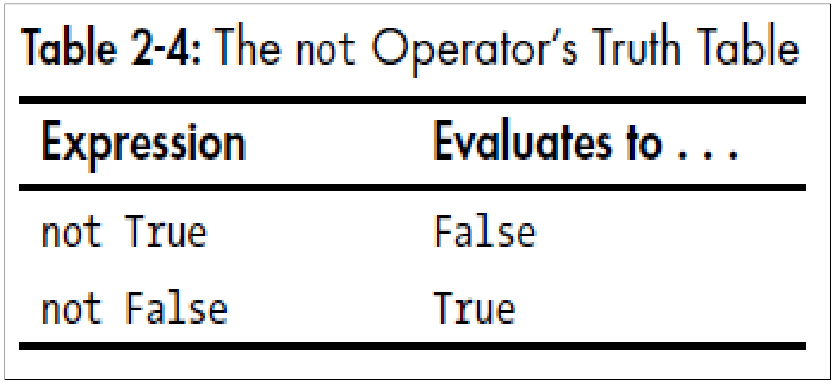
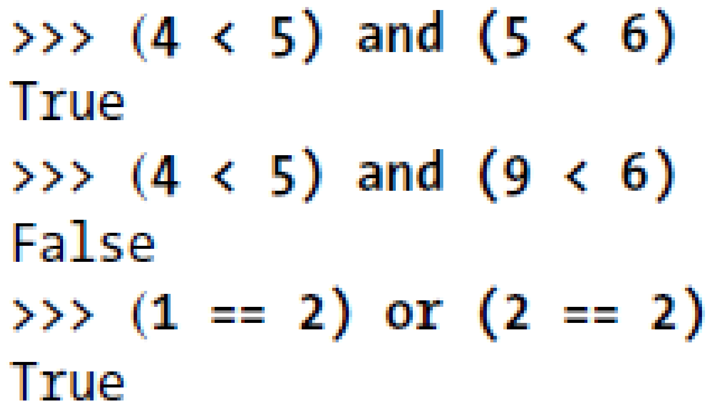


Table 2-4: The not Operator's Truth Table

Expression	Evaluates to . . .
`not True`	`False`
`not False`	`True`

```
>>> not True
False
```

Mixing Boolean and Comparison Operators



```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

Elements of Flow Control

➢***Conditions***

❑Conditions always evaluate down to a Boolean value, True or False.

❑A flow control statement decides what to do based on whether its condition is True or False, and almost every flow control statement uses a condition.

➢***Blocks of Code***

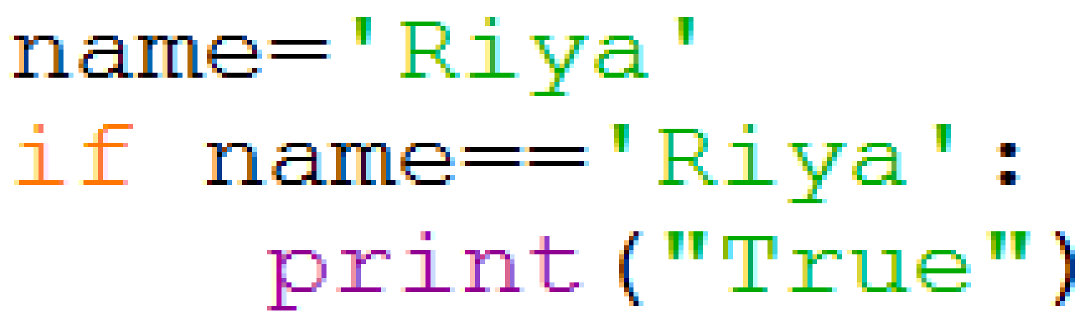
There are three rules for blocks.

- Blocks begin when the indentation increases.
- Blocks can contain other blocks.
- Blocks end when the indentation

Flow Control Statements

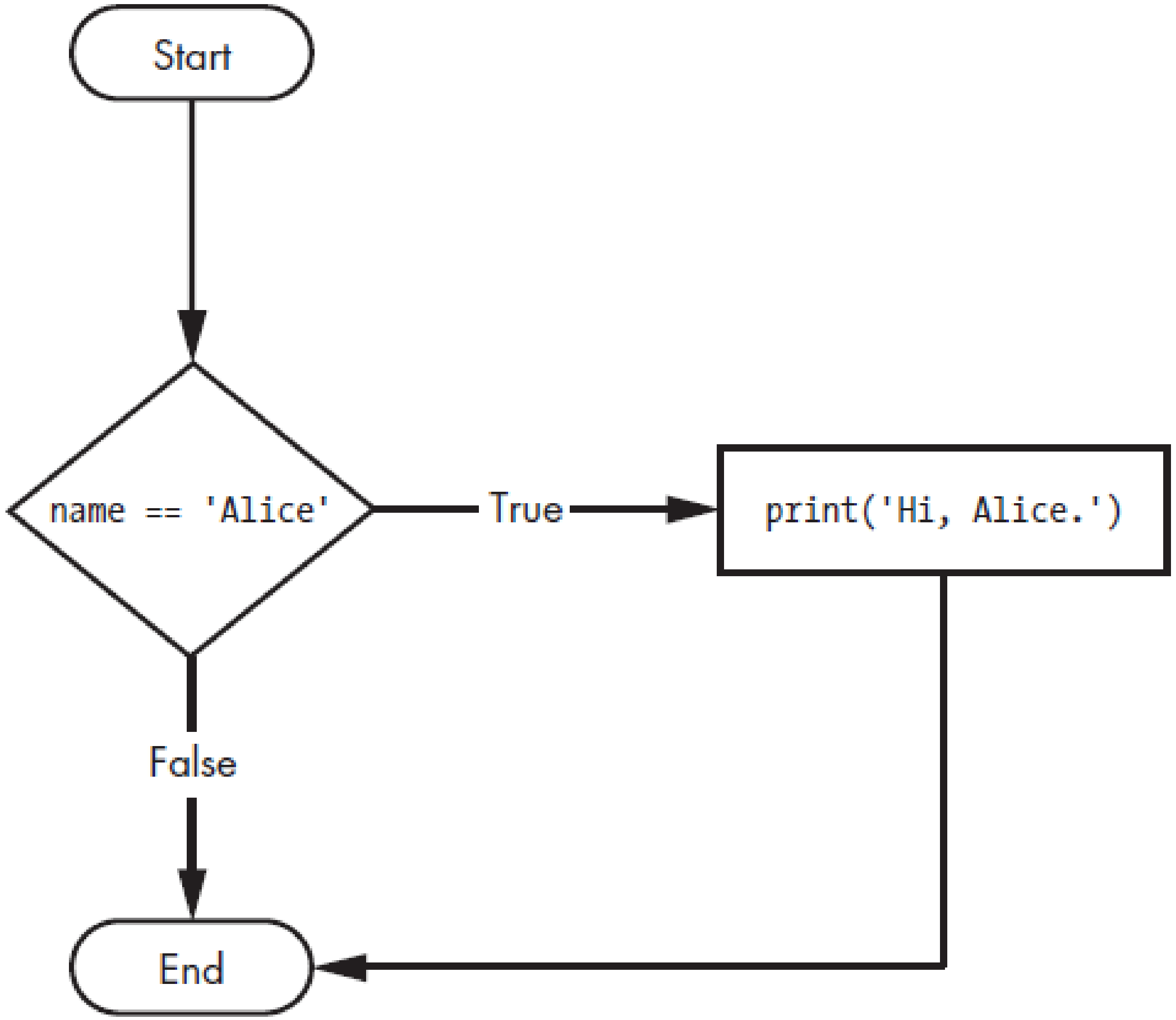
➢ ***if Statements***

- The if keyword
- A condition that is, an expression that evaluates to True or False
- A colon
- Starting on the next line, an indented block of the if



```
name='Riya'
if name=='Riya':
    print("True")
```

Flow Control Statements



Flow Control Statements

else Statements

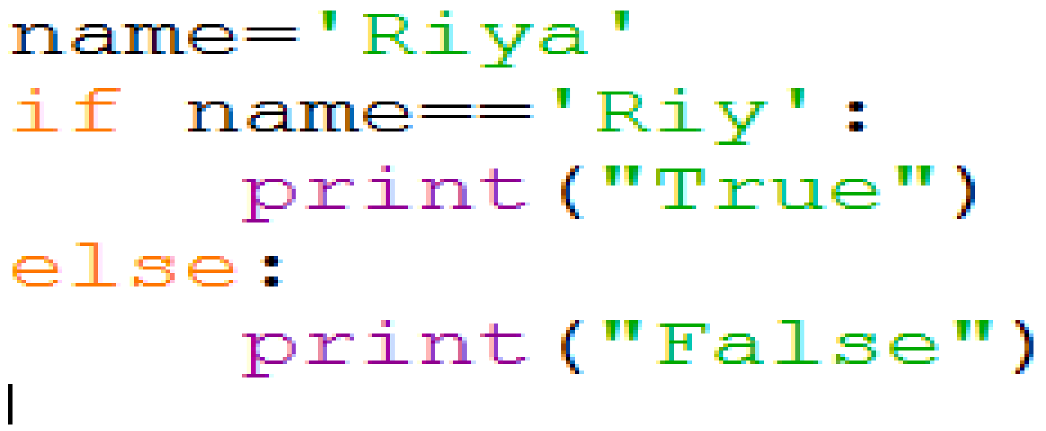
❑"If this condition is true, execute this code. Or else, execute that code."

❑An else statement doesn't have a condition, consists of the following:

- The else keyword
- A colon
- Starting on the next line, an indented block of code (called the else clause)

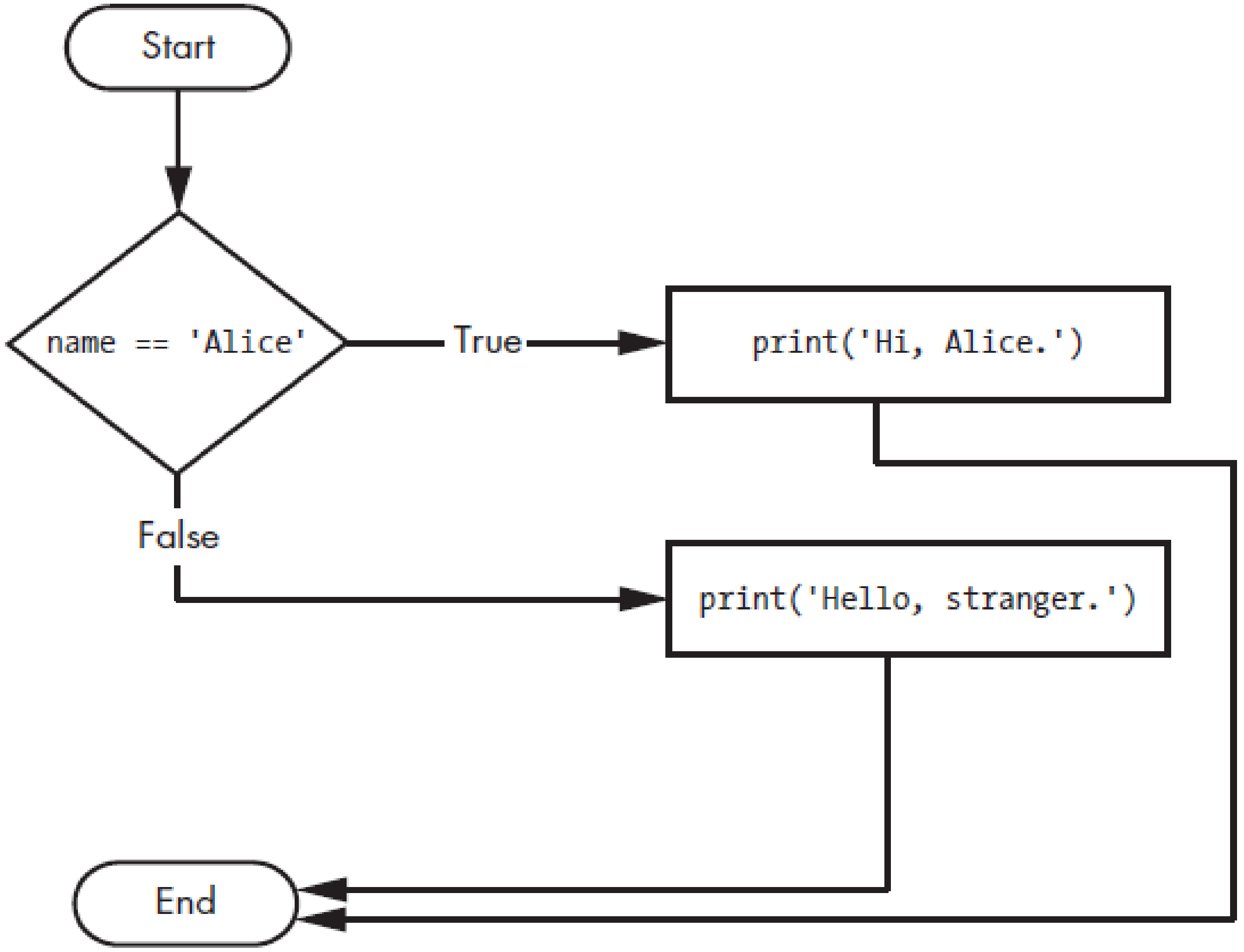
Flow Control Statements

else Statements



```
name='Riya'
if name=='Riy':
    print("True")
else:
    print("False")
```

Flow Control Statements



Flow Control Statements

elif Statements

❑you may have a case where you want one of *many*
possible clauses to execute.

❑elif statement always consists of the following:

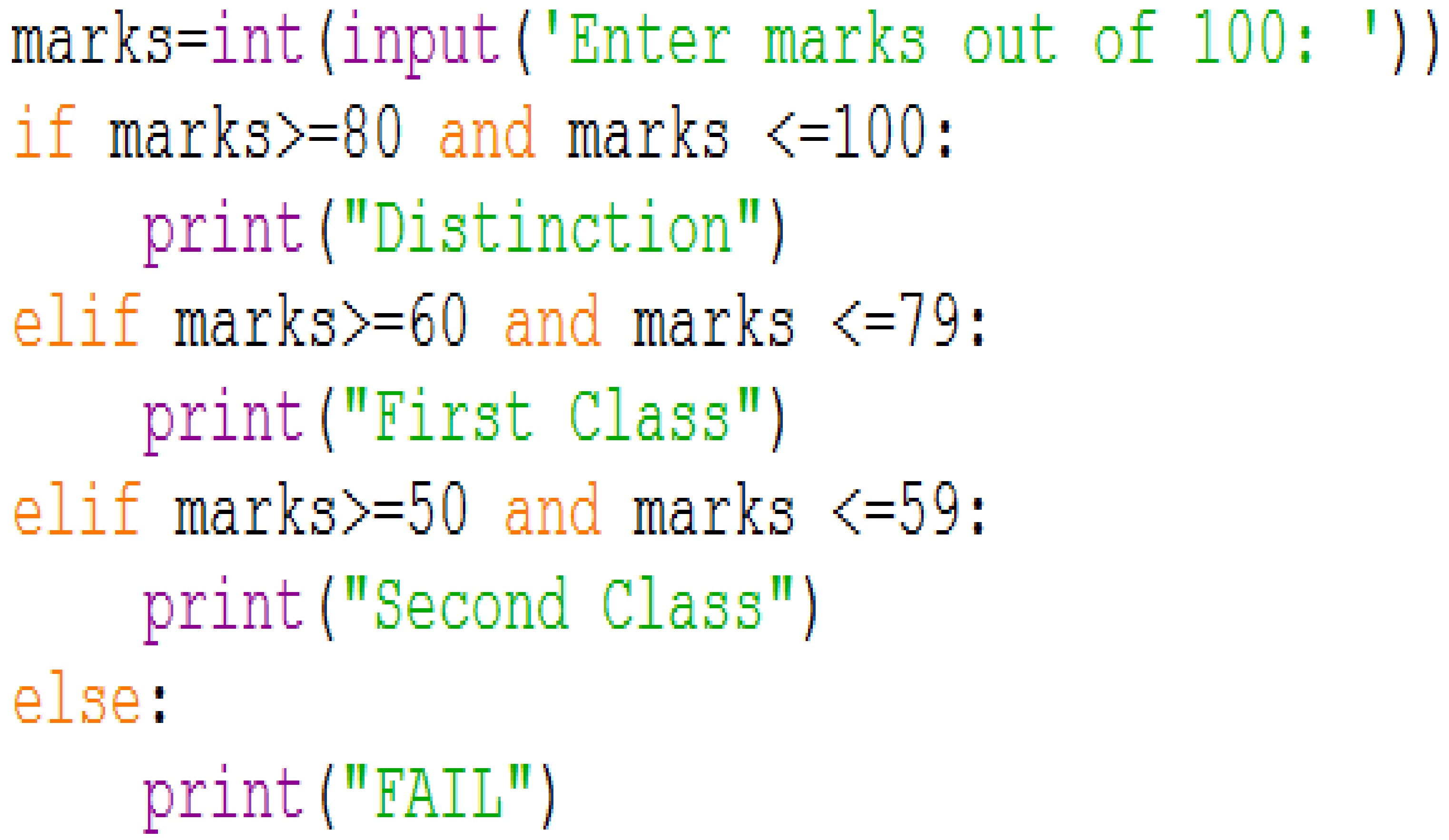
•The elif keyword

• A condition (that is, an expression that evaluates to True or False)

• A colon

• Starting on the next line, an indented block of code (called the **elif**

Flow Control Statements

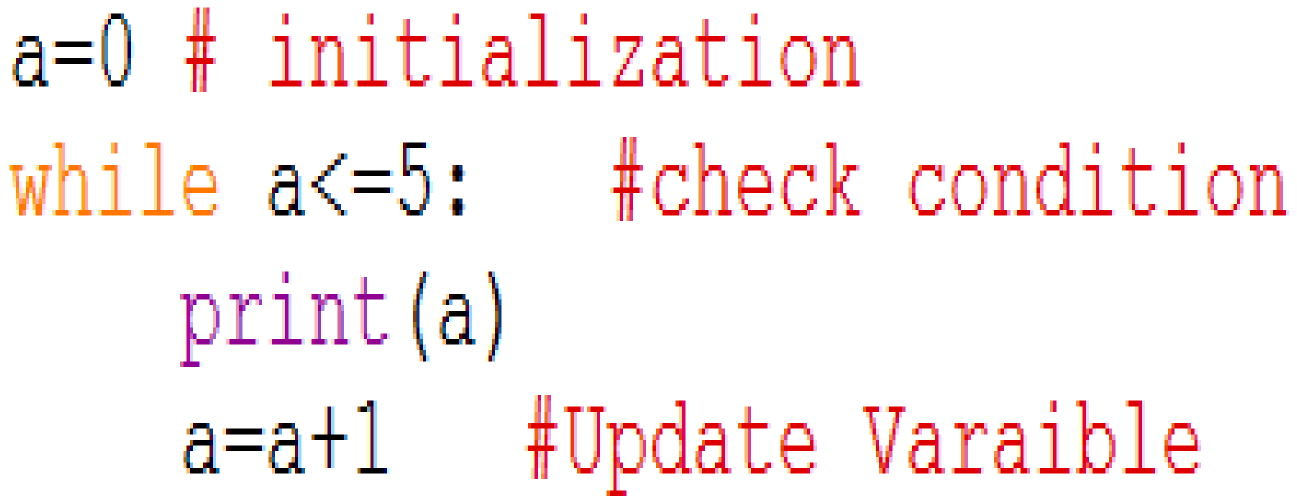


```python
marks=int(input('Enter marks out of 100: '))
if marks>=80 and marks <=100:
    print("Distinction")
elif marks>=60 and marks <=79:
    print("First Class")
elif marks>=50 and marks <=59:
    print("Second Class")
else:
    print("FAIL")
```

while Loop Statements

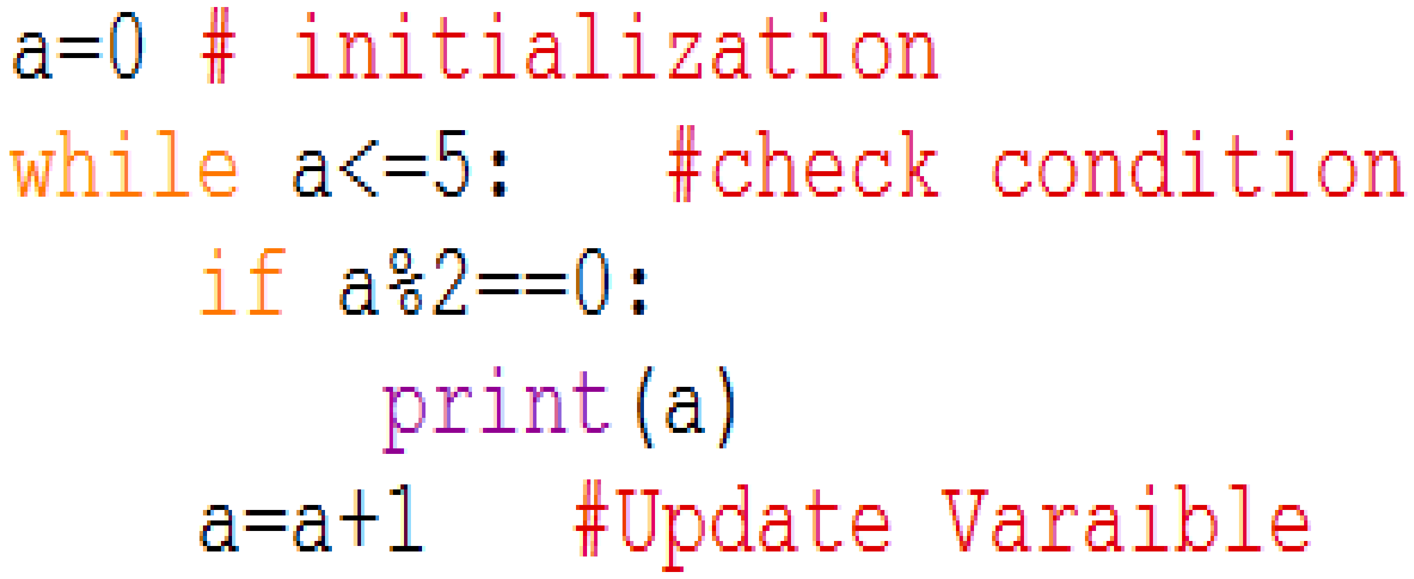
- ➢You can make a block of code execute over and over again using a while statement.
  - The while keyword
  - A condition
  - A colon
  - Starting on the next line, an indented block of code (called the
    while clause)
- ➢At the end of a while clause, the program execution jumps back to the start of the while statement.
- ➢The while clause is often called the while loop or just the loop.

while Loop Statements



```
a=0 # initialization
while a<=5:   #check condition
    print(a)
    a=a+1   #Update Varaible
```

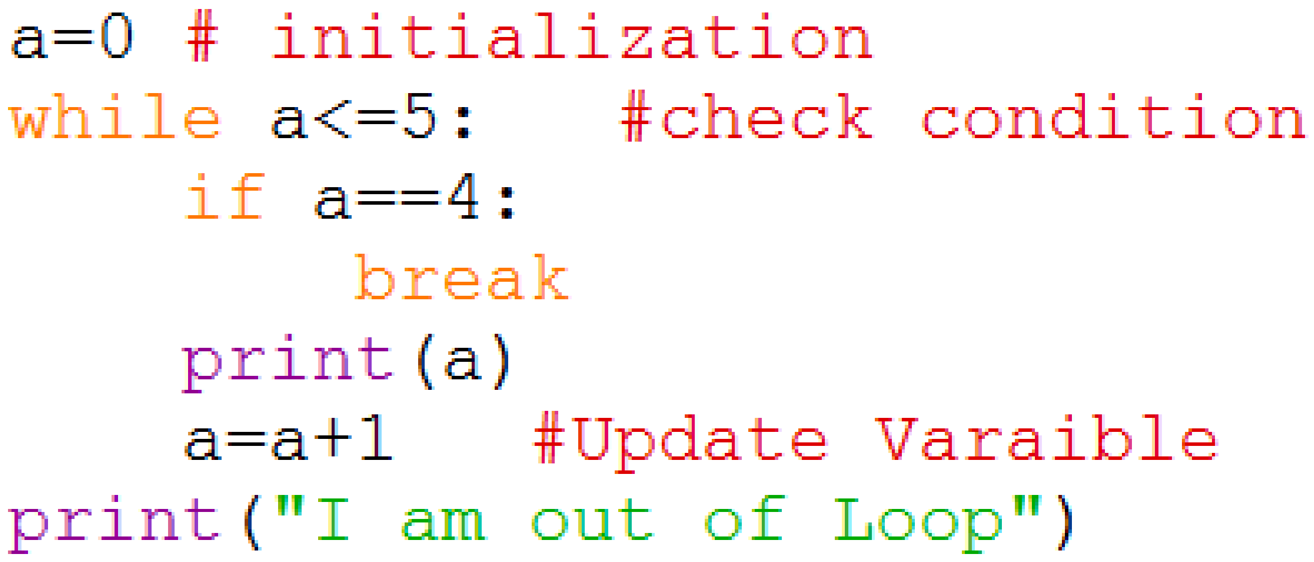
Composition



```
a=0 # initialization
while a<=5:   #check condition
    if a%2==0:
        print(a)
    a=a+1   #Update Varaible
```

break Statements

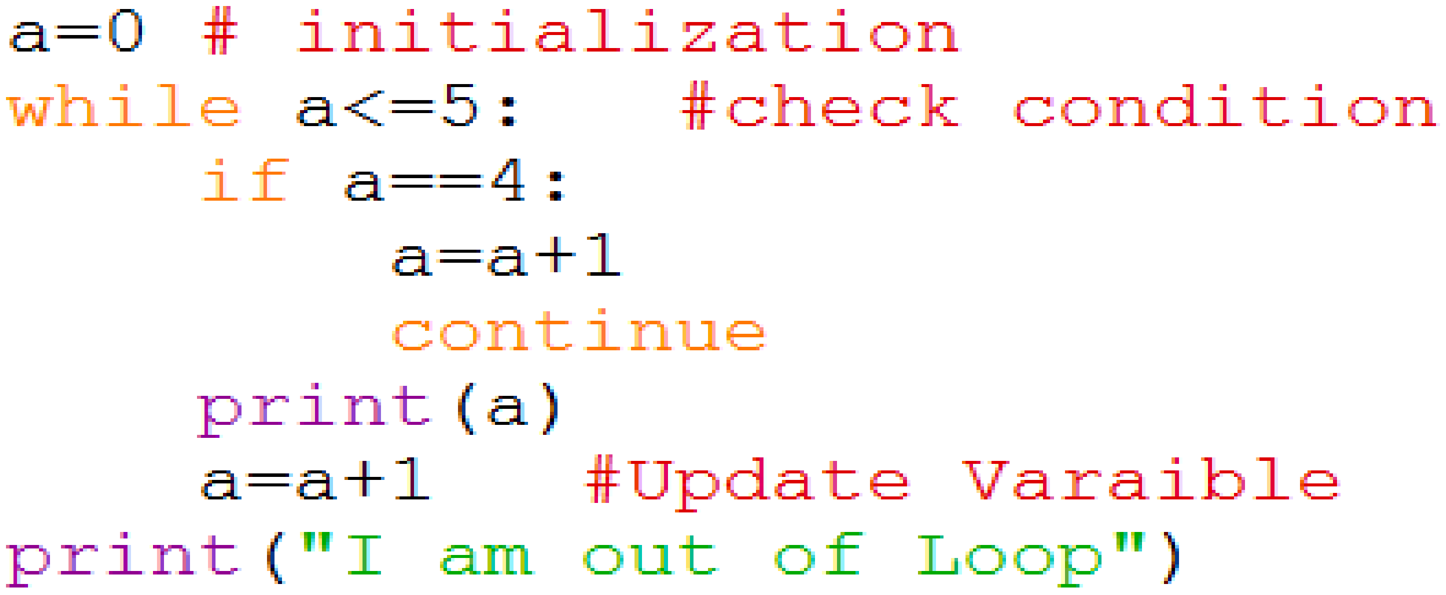
If the execution reaches a break statement, it immediately exits the while loop’s clause.



```
a=0 # initialization
while a<=5:    #check condition
    if a==4:
        break
    print(a)
    a=a+1   #Update Varaible
print("I am out of Loop")
```

continue Statements

When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and revaluates the loop’s condition.



```
a=0 # initialization
while a<=5:   #check condition
    if a==4:
        a=a+1
        continue
    print(a)
    a=a+1   #Update Varaible
print("I am out of Loop")
```

List of Program using while loop

1. Program to print numbers 1 to 10
2. Program to print even numbers between 1 to 10
3. Program to print odd numbers between 1 to 10
4. Demonstrate break statement
5. Demonstrate continue statement

for Loops and the range() Function

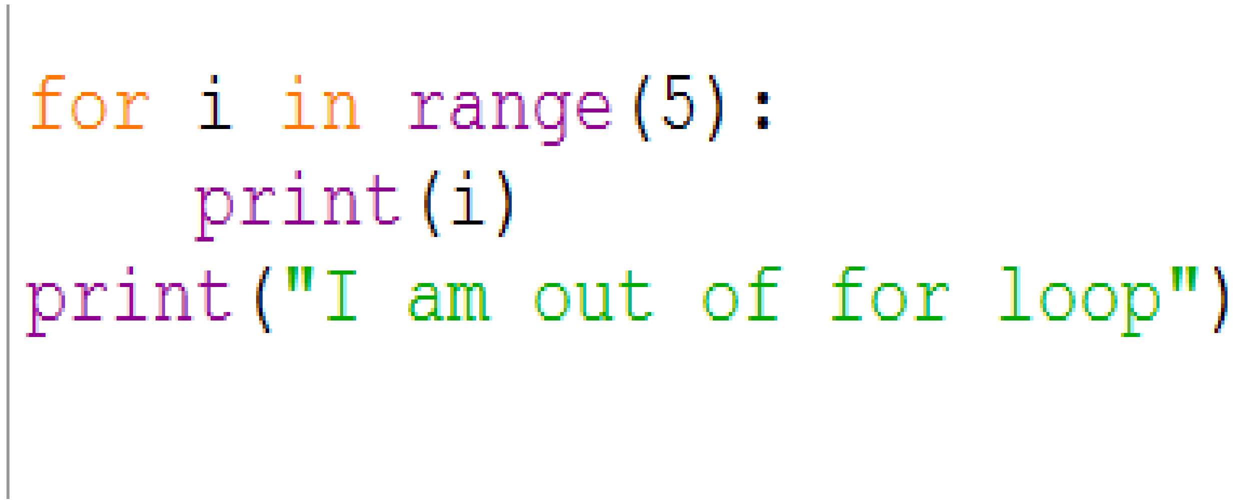
In code, a for statement looks something like

Eg **for i in range(5):**

and includes the following:

- The for keyword
- A variable name
- The in keyword
- A call to the range() method with up to three integers passed to it
- A colon
- Starting on the next line, an indented block of code (called the for clause)

for Loops and the range() Function



```
for i in range(5):
    print(i)
print("I am out of for loop")
```

range() Function

Syntax

range(*start*, *stop*, *step*)

*start*Optional. An integer number specifying at which position to start. Default is 0

*stop*Required. An integer number specifying at which position to stop (not included).

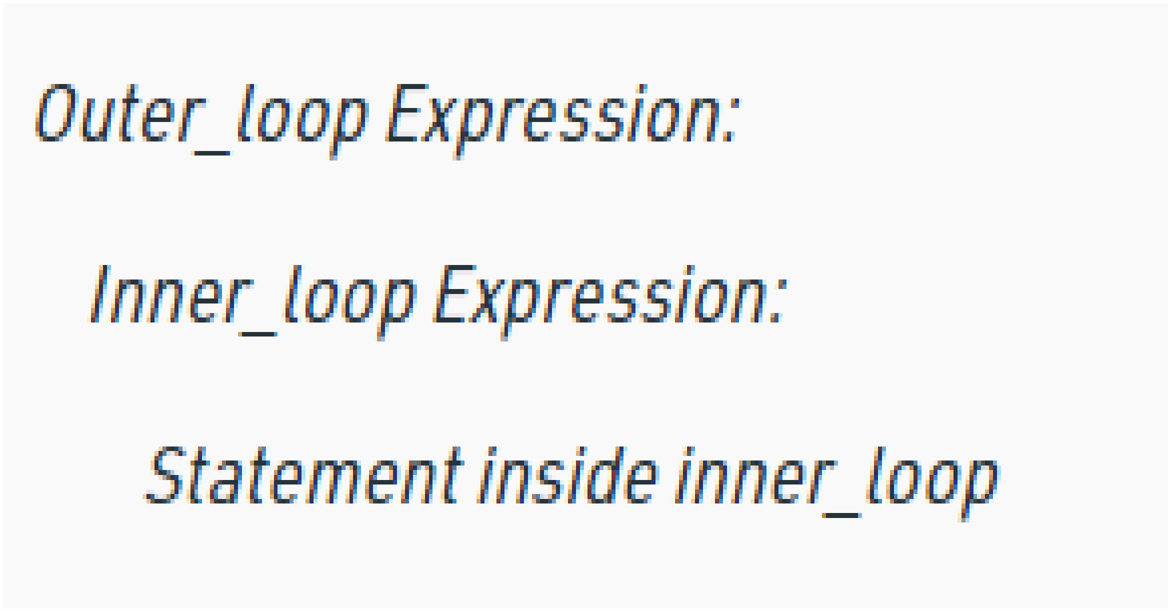
*step*Optional. An integer number

range() Function

```
for n in range(3, 6):
  print(n)
print("I am out of for loop")
```

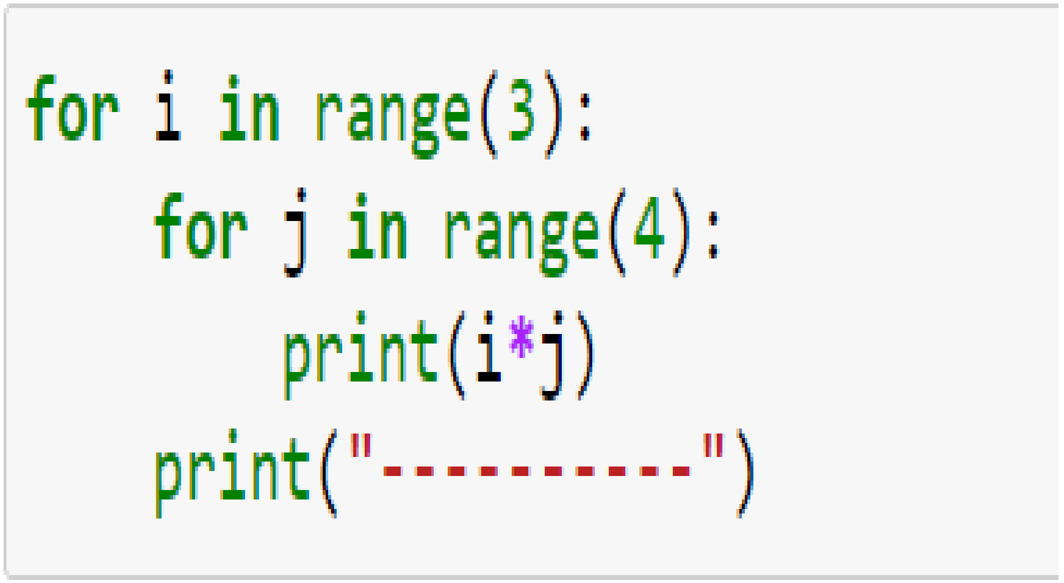
```
for n in range(3, 20, 2):
  print(n)
print("I am out of for loop")
```

Python Nested Loops Syntax:



```
Outer_loop Expression:
    Inner_loop Expression:
        Statement inside inner_loop
```

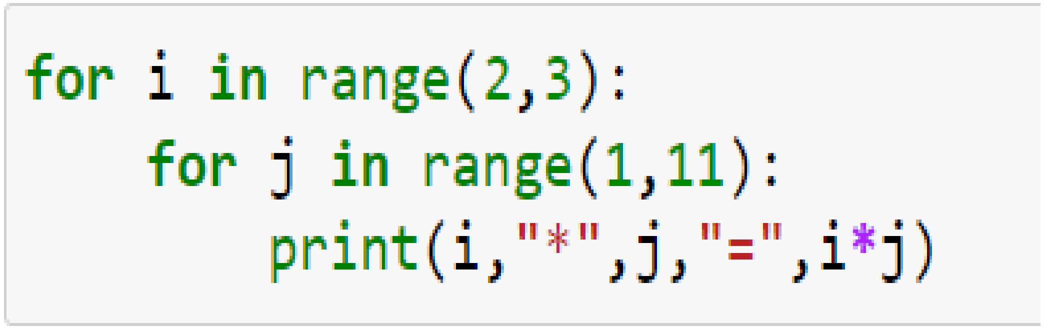
Python Nested Loops Syntax:



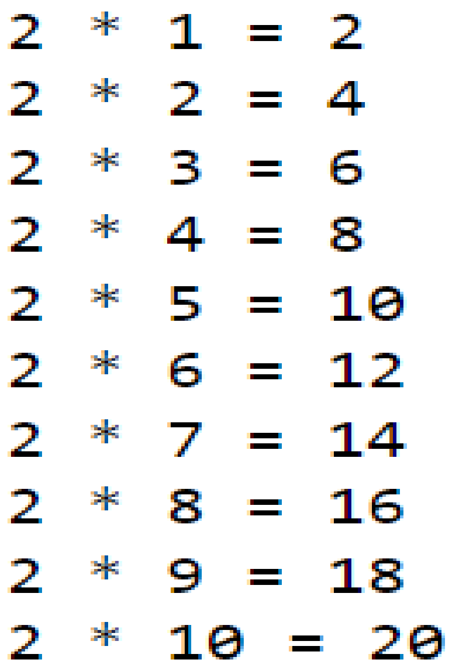
```python
for i in range(3):
    for j in range(4):
        print(i*j)
    print("----------")
```

```
0
0
0
0
----------
0
1
2
3
----------
0
2
4
6
----------
```

Printing multiplication table using Python nested for loops



```python
for i in range(2,3):
    for j in range(1,11):
        print(i,"*",j,"=",i*j)
```



```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

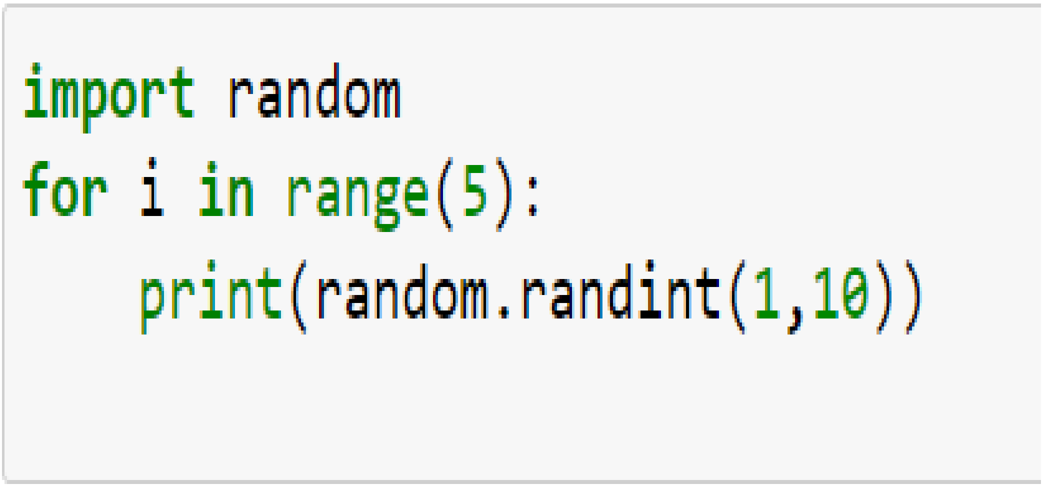
Importing Modules

- *Built-in functions*→print(), input(), and len() functions
- Python also comes with a set of modules called the *standard library*.
- Each **module is a Python** program that contains a related group of functions that can be embedded in your programs.
- Before you can use the functions in a module, you must **import the module** with an import statement.

Importing Modules

- In code, an import statement consists of the following:
- The import keyword
- The name of the module
- Optionally, more module names, as long as they are separated by commas

Eg



```
import random
for i in range(5):
    print(random.randint(1,10))
```

```
6
7
6
2
9
```

Importing Modules

```
import random, sys, os, math
```

from import Statements

from keyword, followed by the module name, the import keyword, and a star;

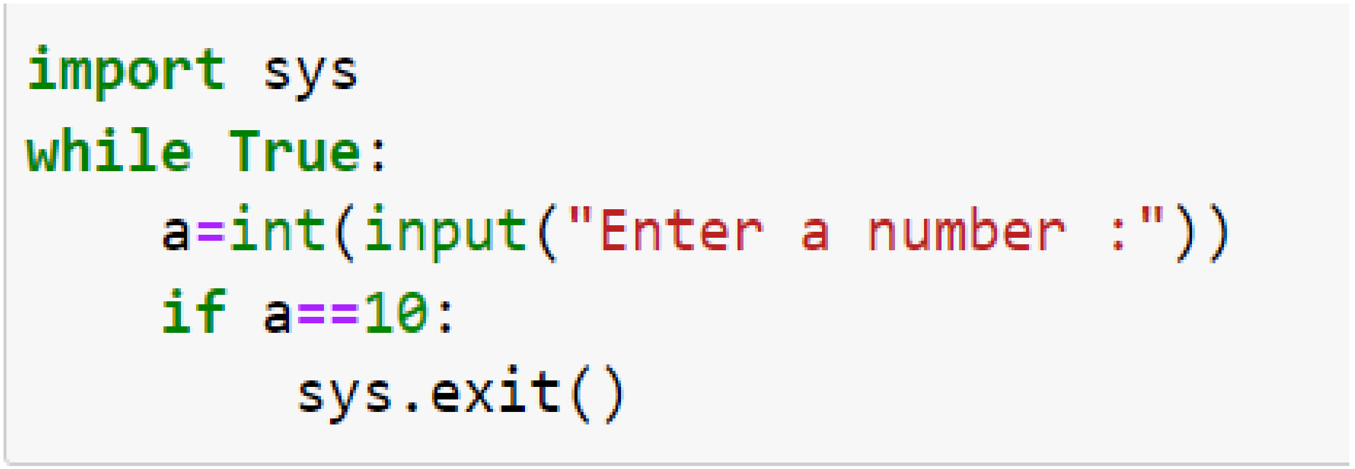
For example,
from random import *.

Ending a Program Early with the sys.exit() Function

How to terminate the program ?

- Programs always terminate if the program execution reaches the bottom of the instructions
- you can cause the program to terminate, or exit, before the last instruction by calling the **sys.exit() function**
- Since this function is in the sys module, you have to import sys before your program can use it.

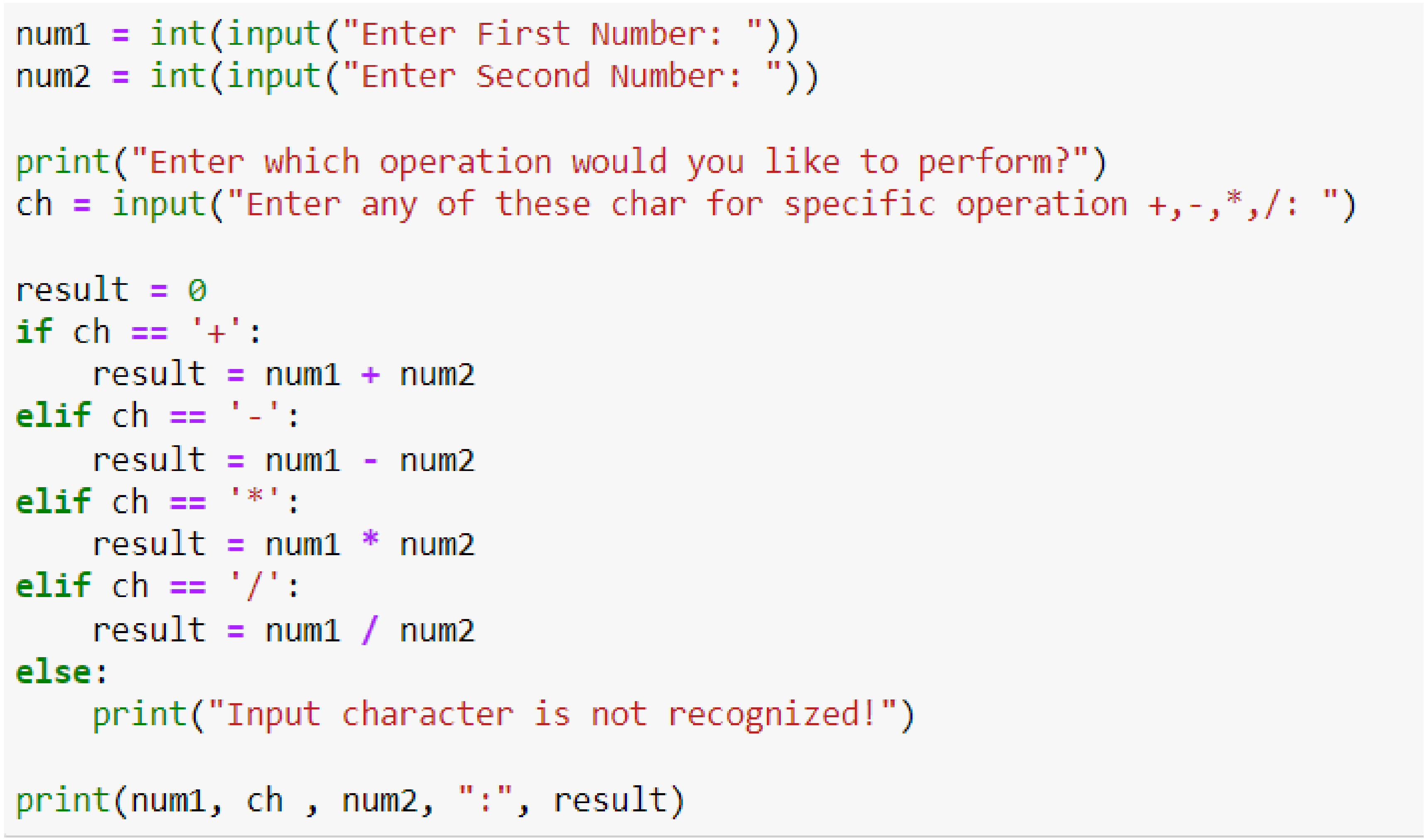
Ending a Program Early with the sys.exit() Function



```
import sys
while True:
    a=int(input("Enter a number :"))
    if a==10:
        sys.exit()
```

```
Enter a number :5
Enter a number :8
Enter a number :10

An exception has occurred, use %tb to see the full traceback.

SystemExit
```

Lab Program 1 : Program to perform addition, subtraction, multiplication and division on two input numbers in Python



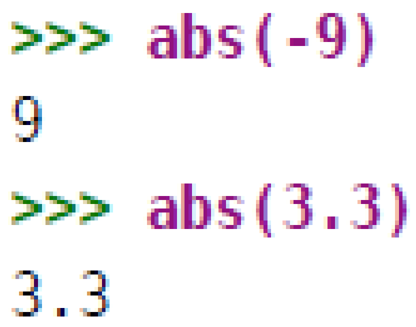
```python
num1 = int(input("Enter First Number: "))
num2 = int(input("Enter Second Number: "))

print("Enter which operation would you like to perform?")
ch = input("Enter any of these char for specific operation +,-,*,/: ")

result = 0
if ch == '+':
    result = num1 + num2
elif ch == '-':
    result = num1 - num2
elif ch == '*':
    result = num1 * num2
elif ch == '/':
    result = num1 / num2
else:
    print("Input character is not recognized!")

print(num1, ch , num2, ":", result)
```

Augmented Assignment

a=a+5	a+=5
b =b*2	b*=2
c = c/5	c/=5
d = d -6	d-=6

Functions

- Easy
- Reuse
- *Built-in functions*



```
>>> abs(-9)
9
>>> abs(3.3)
3.3
```

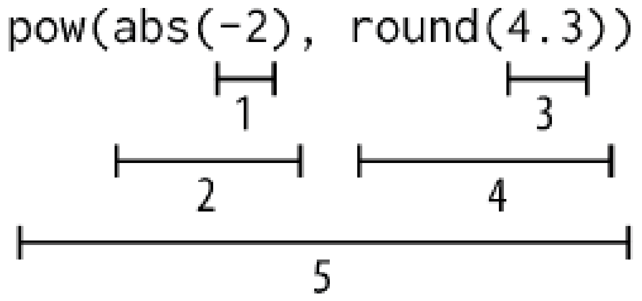
•The general form of a function call is as follows:

«function_name»(«arguments»)

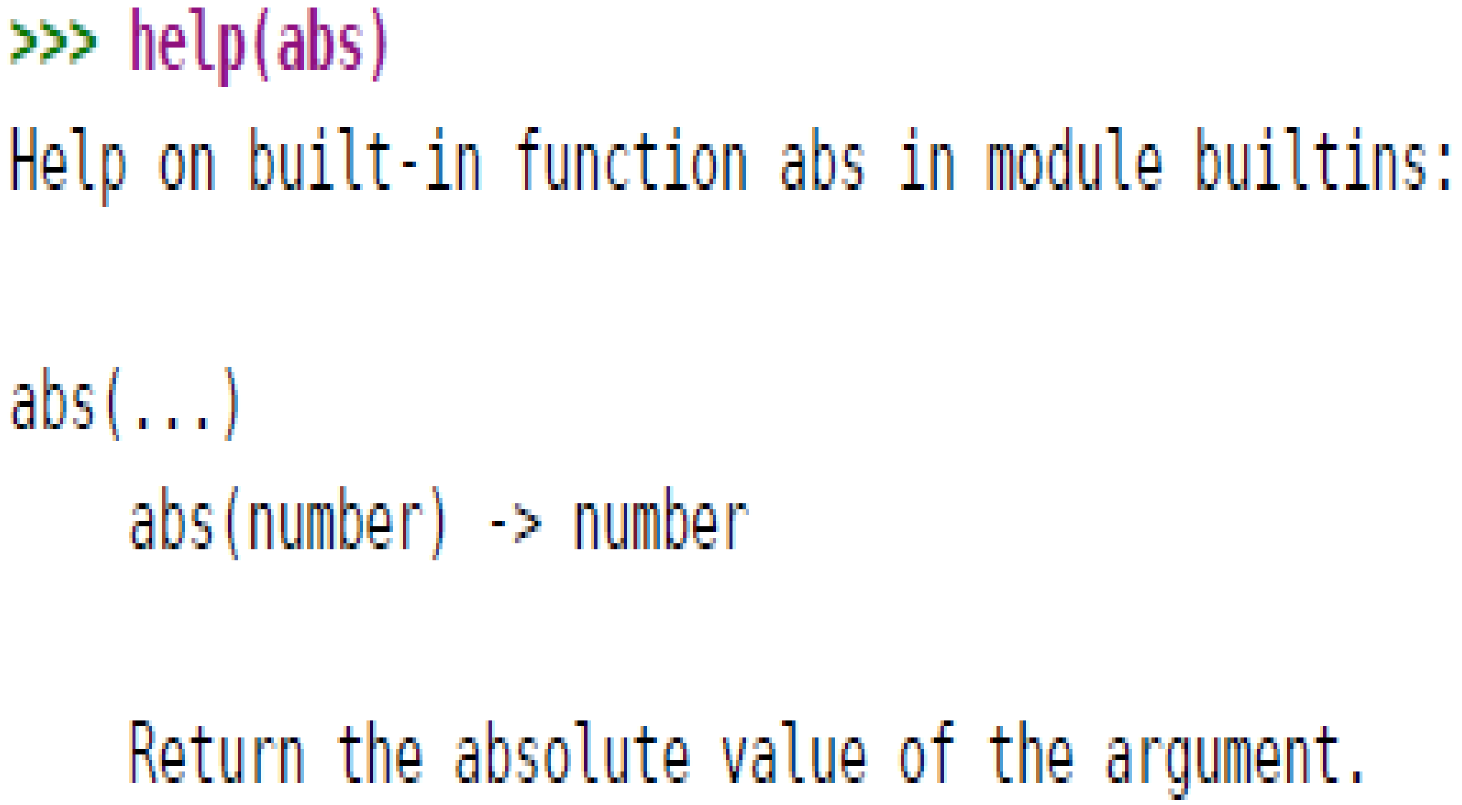
Functions

•Here are the rules to executing a function call:

1. Evaluate each argument one at a time, working from left to right.
2. Pass the resulting values into the function.
3. Execute the function. When the function call finishes, it produces a value.

Eg: **pow(**  **3))**

Functions



```
>>> help(abs)
Help on built-in function abs in module builtins:

abs(...)
    abs(number) -> number

    Return the absolute value of the argument.
```

Functions

```
>>> round(3.8)
4
>>> round(3.3)
3
>>> round(3.5)
4
>>> round(-3.3)
-3
>>> round(-3.5)
-4
>>> round(3.141592653 ,   2)
3.14
```

Functions

```
>>> pow(2, 4)
16
```

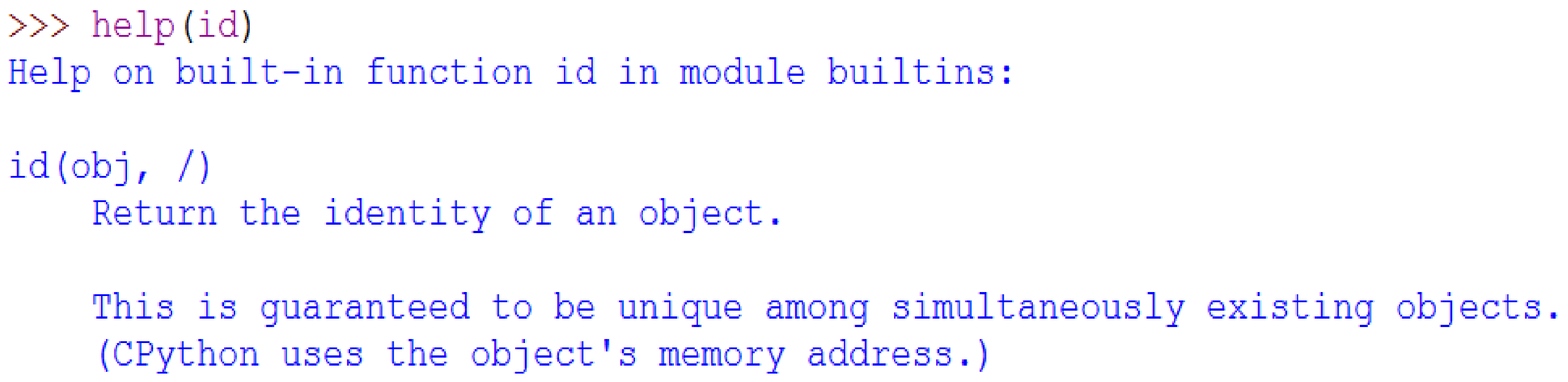
This call calculates 2^4. So far, so good. How about with three arguments?

```
>>> pow(2, 4, 3)
1
```

We know that 2^4 is 16, and evaluation of 16 % 3 produces 1.

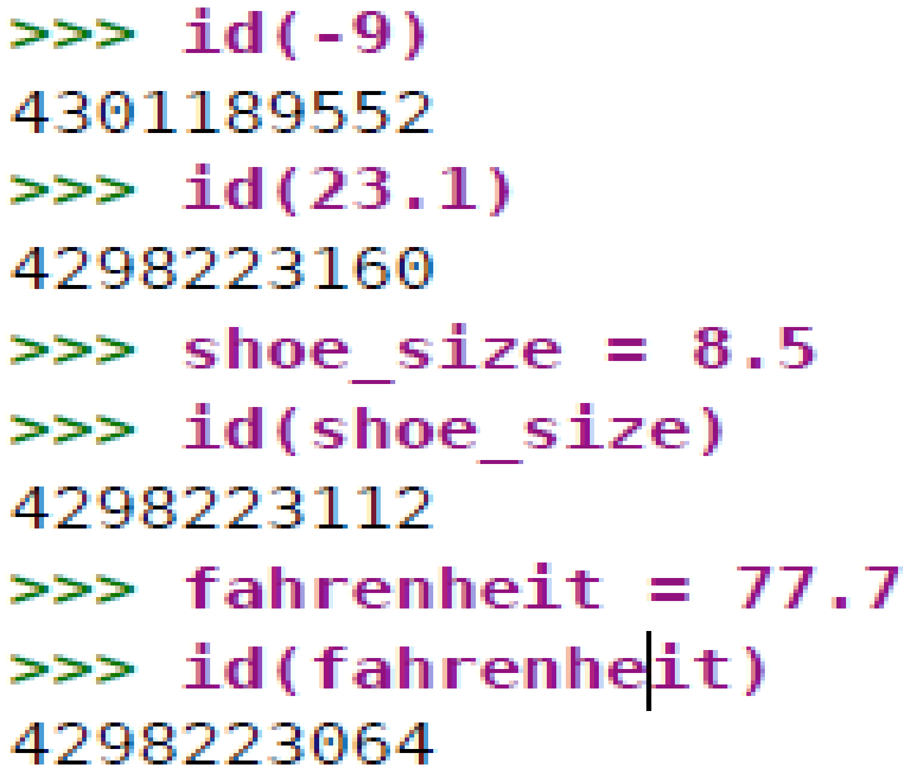
Memory Addresses: How Python Keeps Track of Values

- Python keeps track of each value in a separate object and that each object has a memory address.
- You can discover the actual memory address of an object using built-in function id:



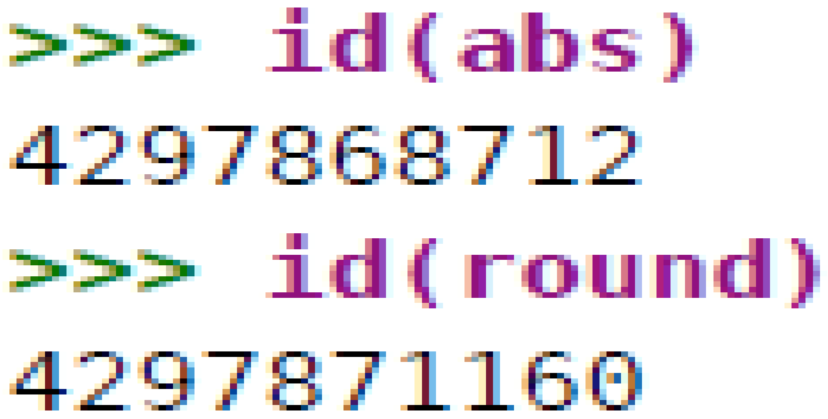
```
>>> help(id)
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```

Memory Addresses



```
>>> id(-9)
4301189552
>>> id(23.1)
4298223160
>>> shoe_size = 8.5
>>> id(shoe_size)
4298223112
>>> fahrenheit = 77.7
>>> id(fahrenheit)
4298223064
```

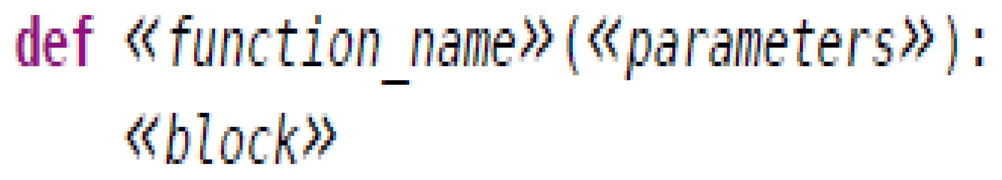
Function also have Memory Addresses



```
>>> id(abs)
4297868712
>>> id(round)
4297871160
```

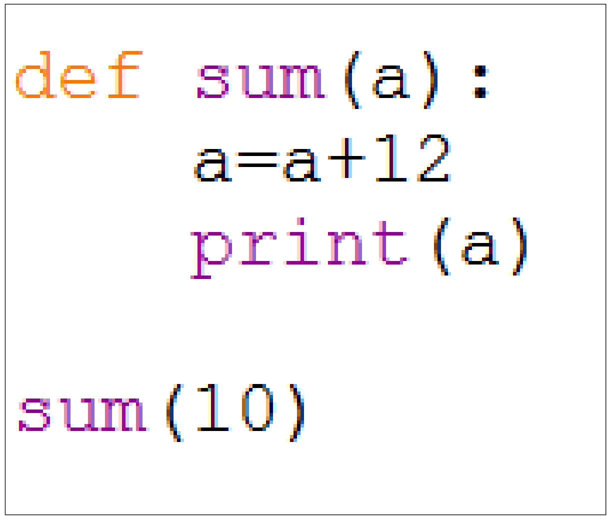
Defining Our Own Functions

- The general form of a function definition is as fc



```
def «function_name»(«parameters»):
    «block»
```

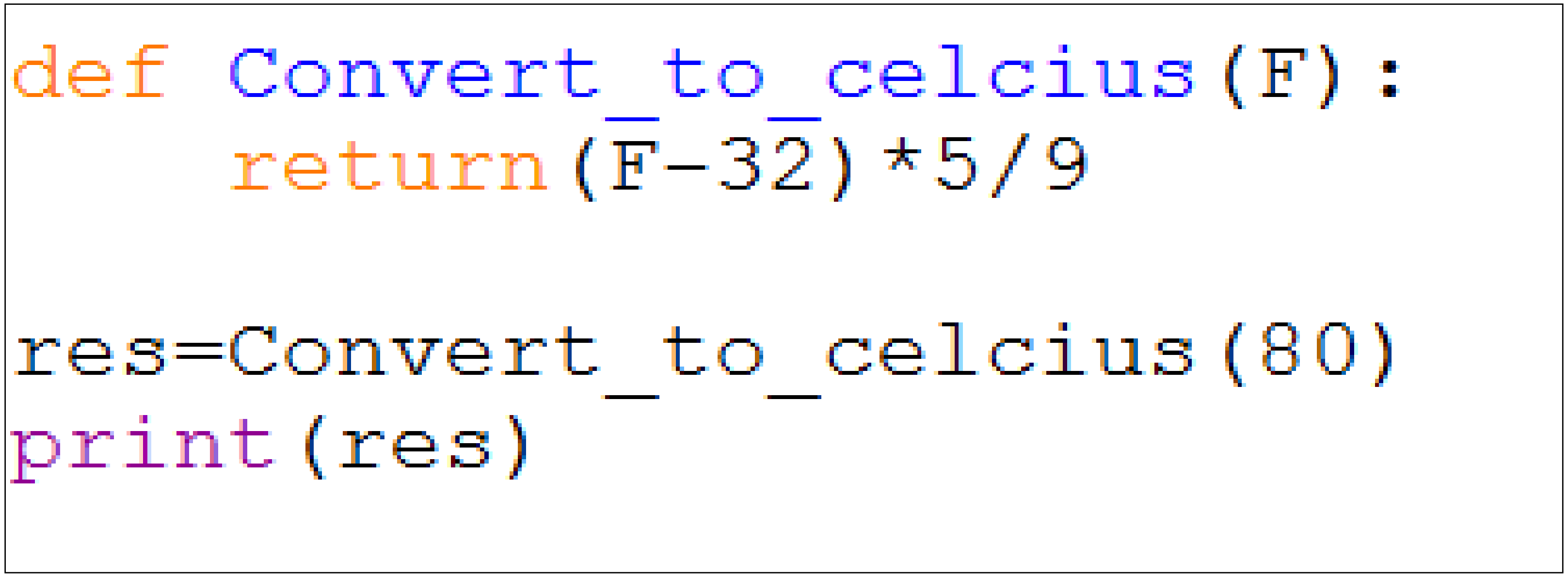
- Example



```
def sum(a):
    a=a+12
    print(a)

sum(10)
```

Defining Our Own Functions

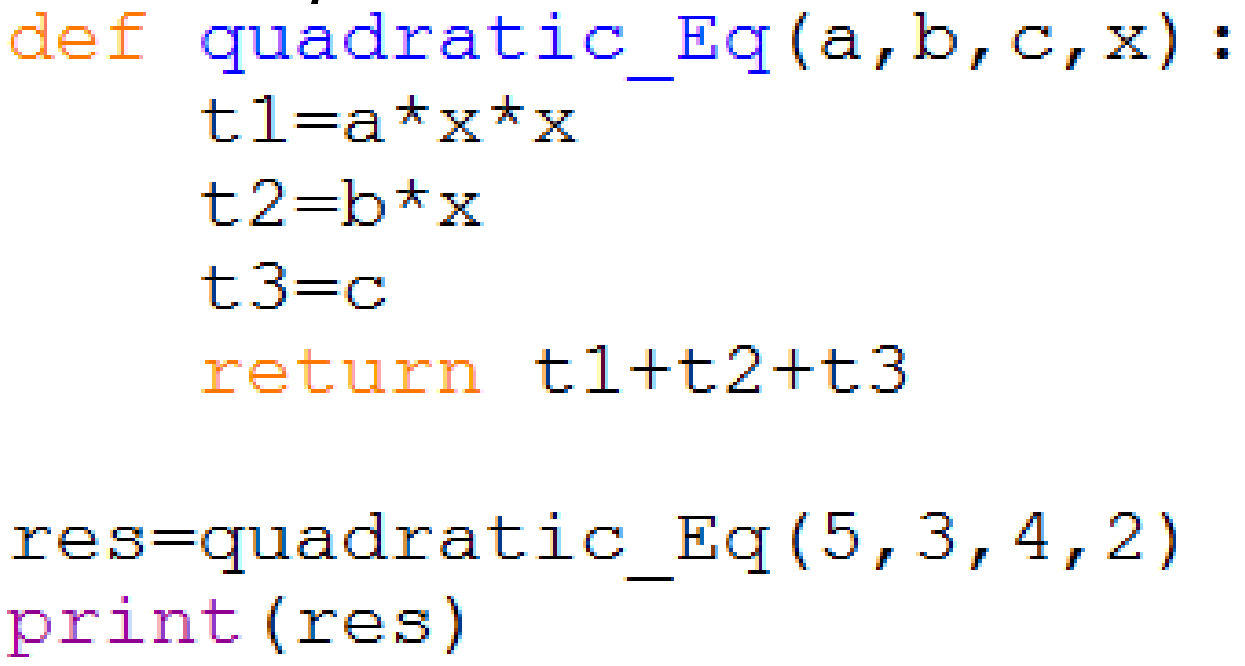
return ***«expression»***



```
def Convert_to_celcius(F):
    return(F-32)*5/9

res=Convert_to_celcius(80)
print(res)
```

Using Local Variables for Temporary Storage

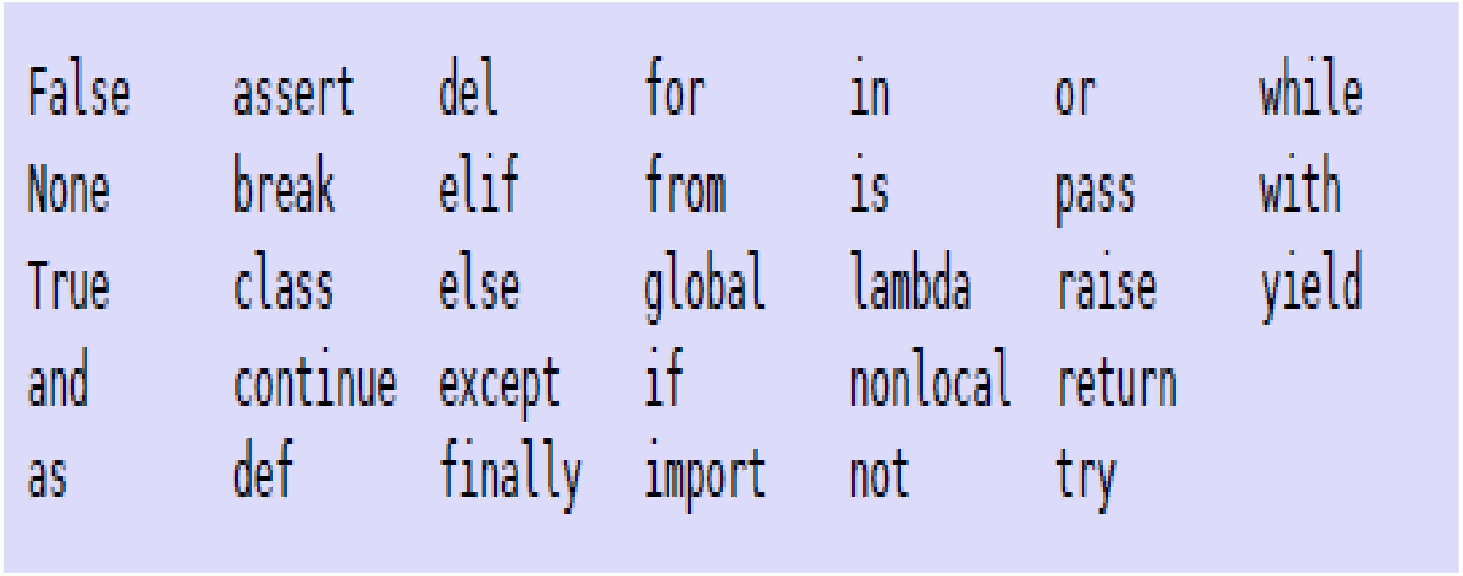
variable's *scope. The scope of a local variable is from the line in which it is defined up* until the end of the function.



```
def quadratic_Eq(a,b,c,x):
    t1=a*x*x
    t2=b*x
    t3=c
    return t1+t2+t3

res=quadratic_Eq(5,3,4,2)
print(res)
```

Keywords

***Keywords** are words that Python reserves for its own use*



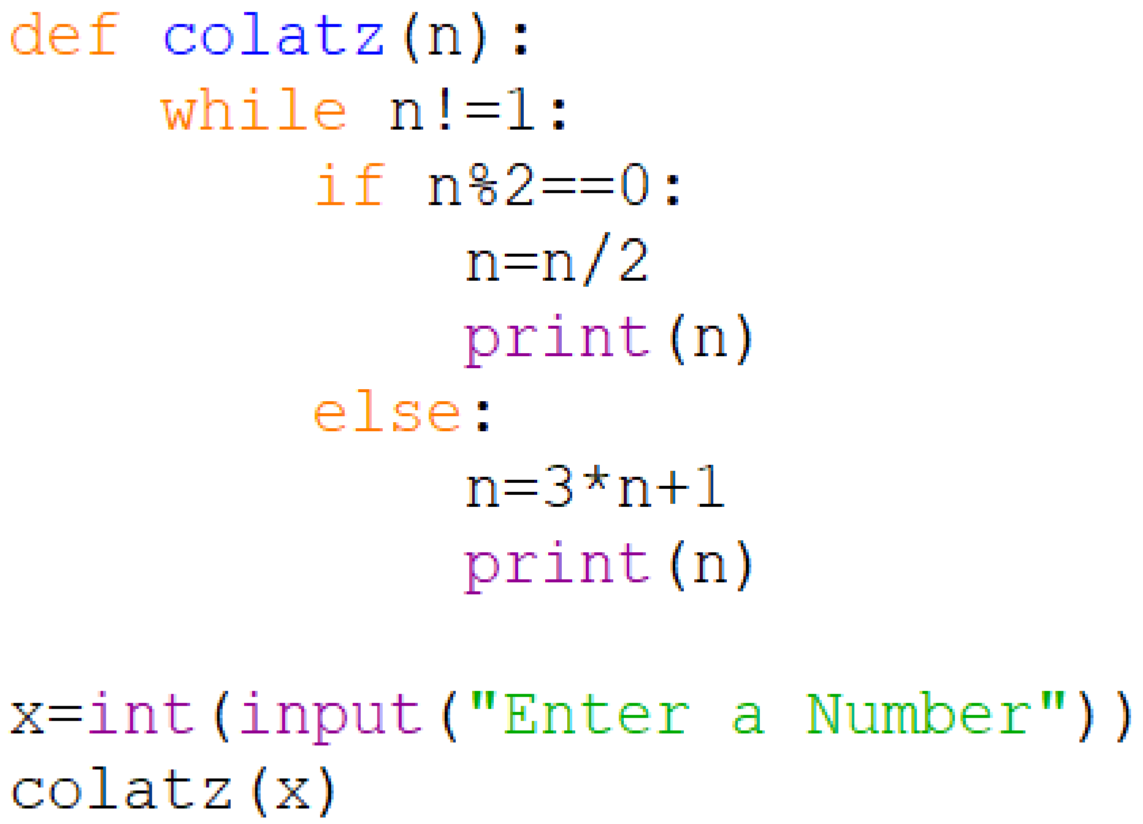
False	assert	del	for	in	or	while
None	break	elif	from	is	pass	with
True	class	else	global	lambda	raise	yield
and	continue	except	if	nonlocal	return	
as	def	finally	import	not	try	

Collatz sequence

The **Collatz sequence** is generated based on the following conditions:

- If the number is *even*, the function returns a value of n//2.
- If the number is *odd*, the function returns the value of 3*number+1.

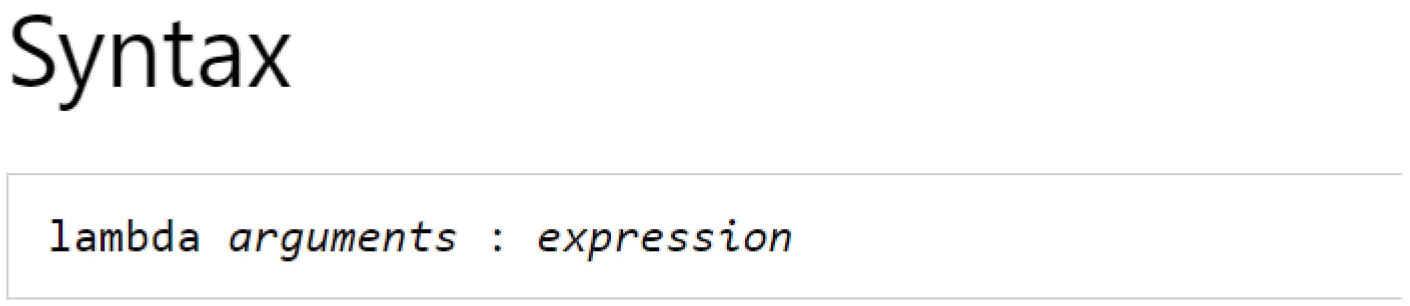
Collatz sequence



```python
def colatz(n):
    while n!=1:
        if n%2==0:
            n=n/2
            print(n)
        else:
            n=3*n+1
            print(n)

x=int(input("Enter a Number"))
colatz(x)
```

Lambda function

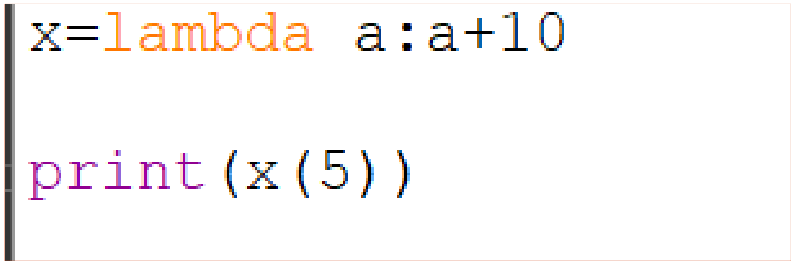
- A lambda function is a small anonymous function.
- lambda function, which allows us to create a one-line function anywhere we want without giving it a name
- A lambda function can take any number of arguments but only have one expression.



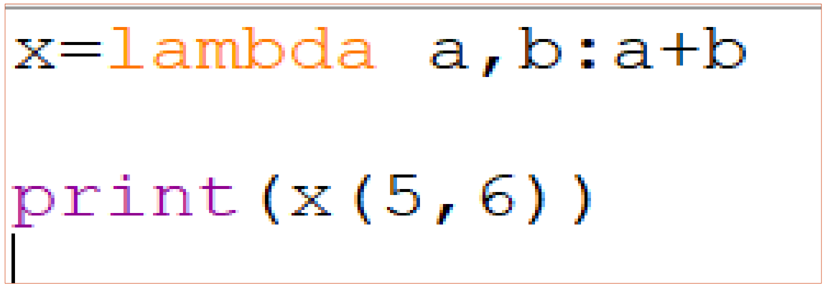
Syntax

```
lambda arguments : expression
```

Lambda function

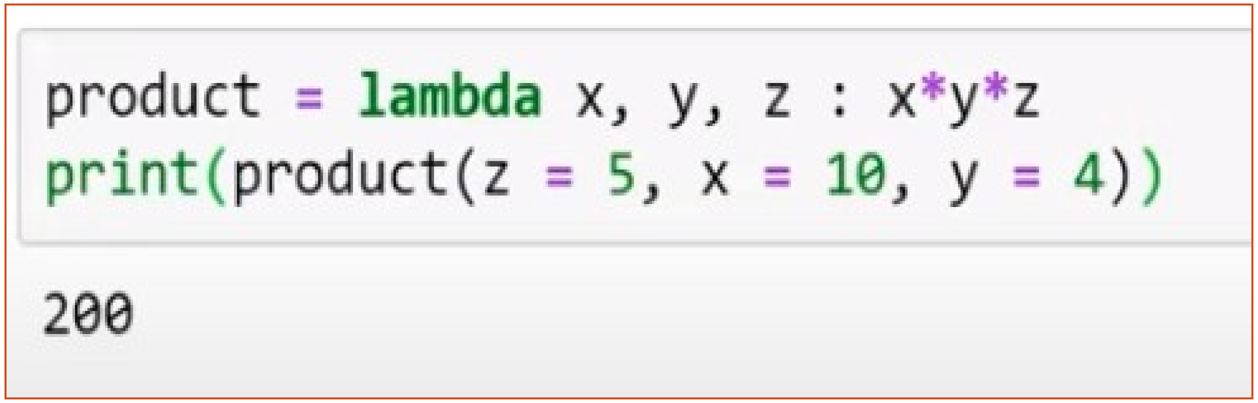


```
x=lambda a:a+10

print(x(5))
```



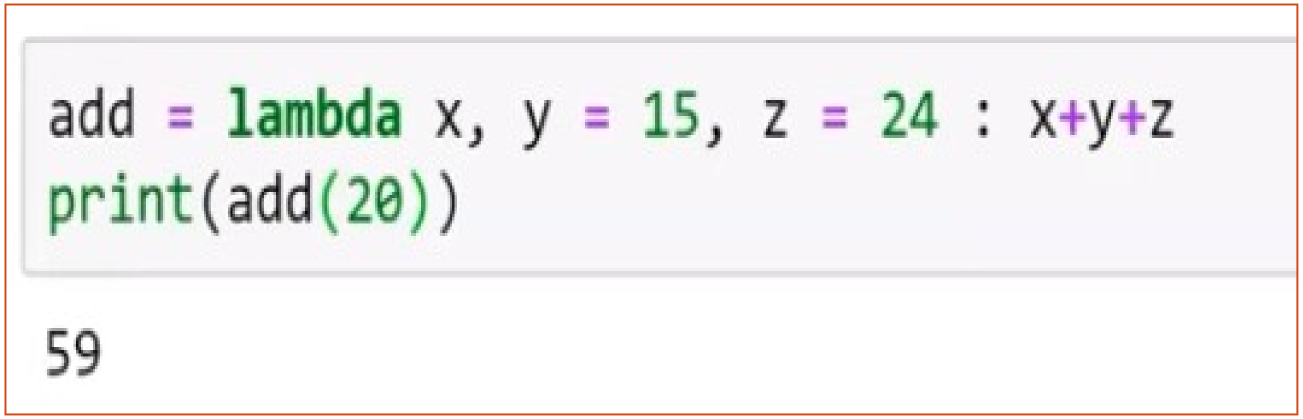
```
x=lambda a,b:a+b

print(x(5,6))
```

Lambda function



```
product = lambda x, y, z : x*y*z
print(product(z = 5, x = 10, y = 4))
```

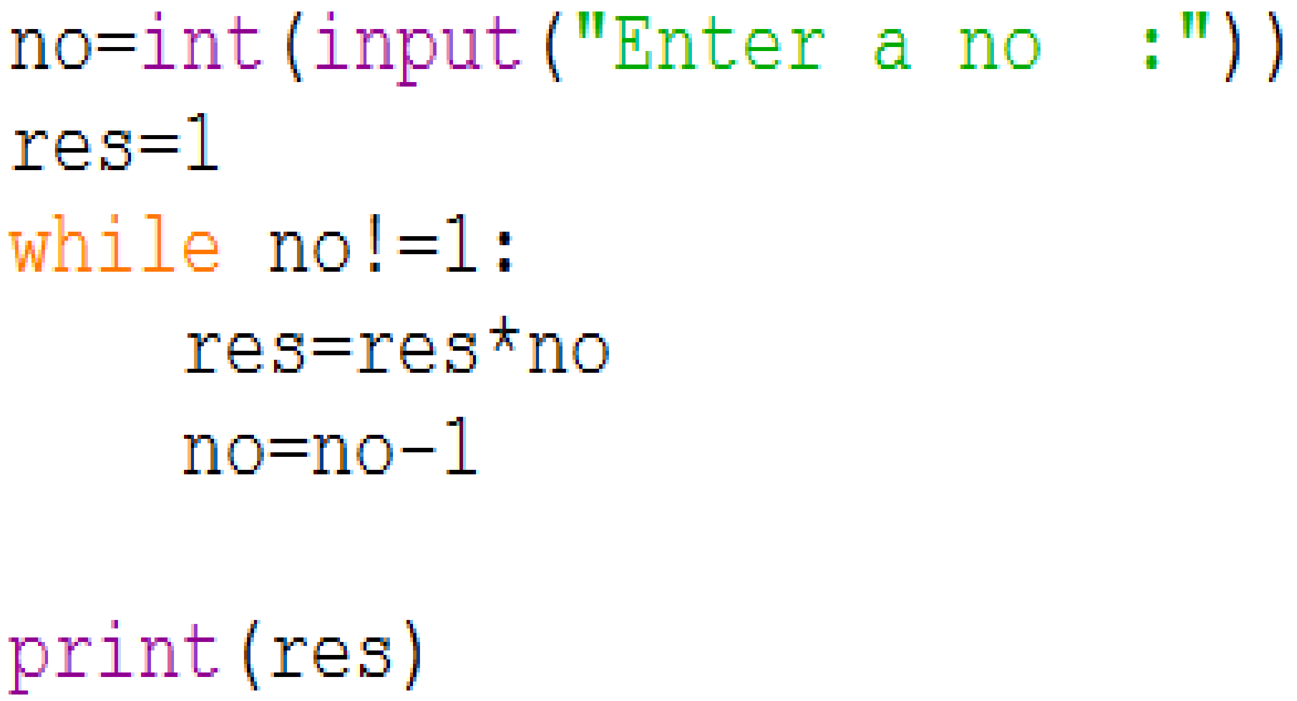
```
200
```



```
add = lambda x, y = 15, z = 24 : x+y+z
print(add(20))
```

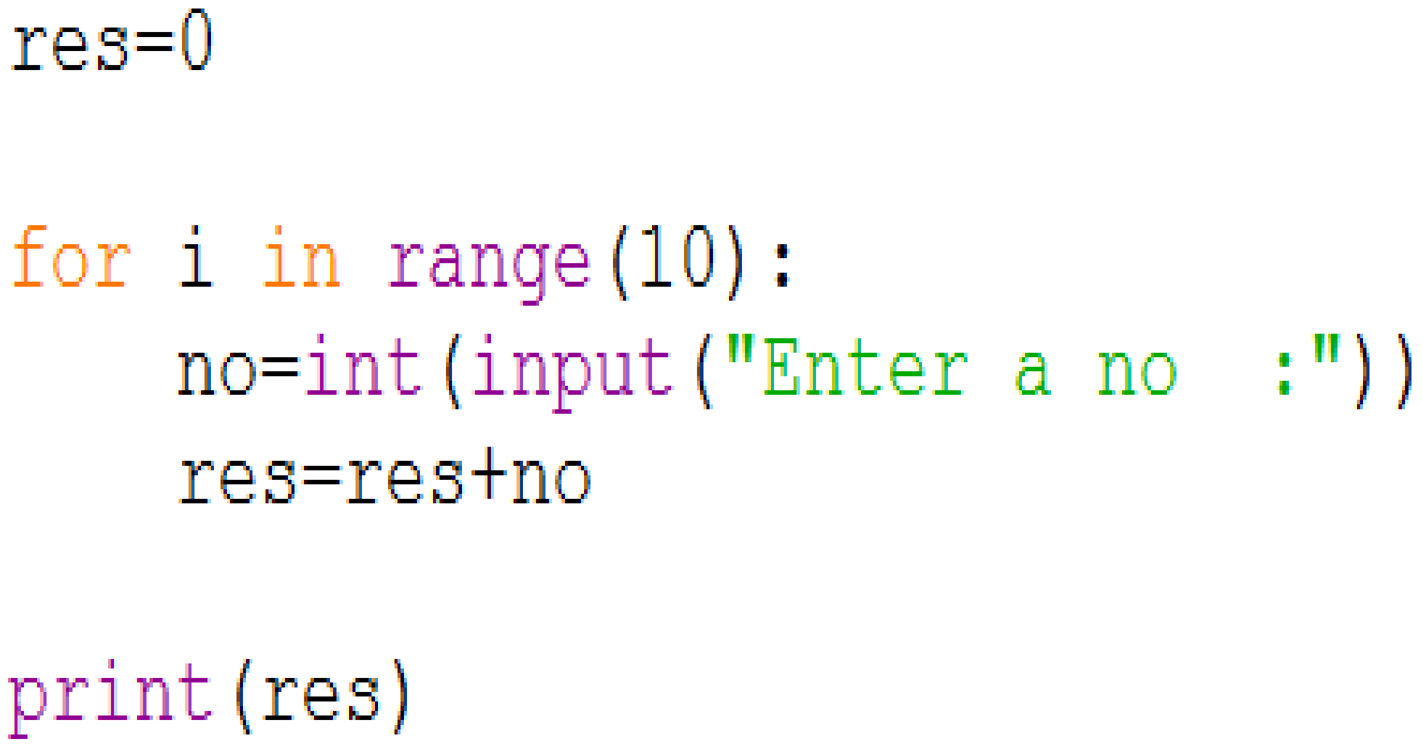
```
59
```

Write a python program to find the factorial of number using while loop.



```python
no=int(input("Enter a no  :"))
res=1
while no!=1:
    res=res*no
    no=no-1

print(res)
```

Write a python program to add 10 numbers by inputting each from the keyboard using for loop.



```python
res=0

for i in range(10):
    no=int(input("Enter a no  :"))
    res=res+no

print(res)
```

2 a] Write a python function linearSearch() to read an array and search for the key element. Display the appropriate messages. Use the recursive function.