

# Python Language



# Content



- Introduction
- IDLE
- Data types
- Mathematical operators
- Input /Output
- Strings
- Range() function
- Control structure
- Iteration
- For loop and while loop

# Introduction to Python



- Python is a powerful modern computer programming language.
- Python allows you to use variables without declaring them (i.e., it determines types implicitly), and it relies on indentation as a control structure.
- Python was developed by Guido van Rossum, and it is free software.
- The object-oriented nature of Python was part of its design from the very beginning.
- Python is available on a wide variety of platforms.

# Introduction to Python (Cont)



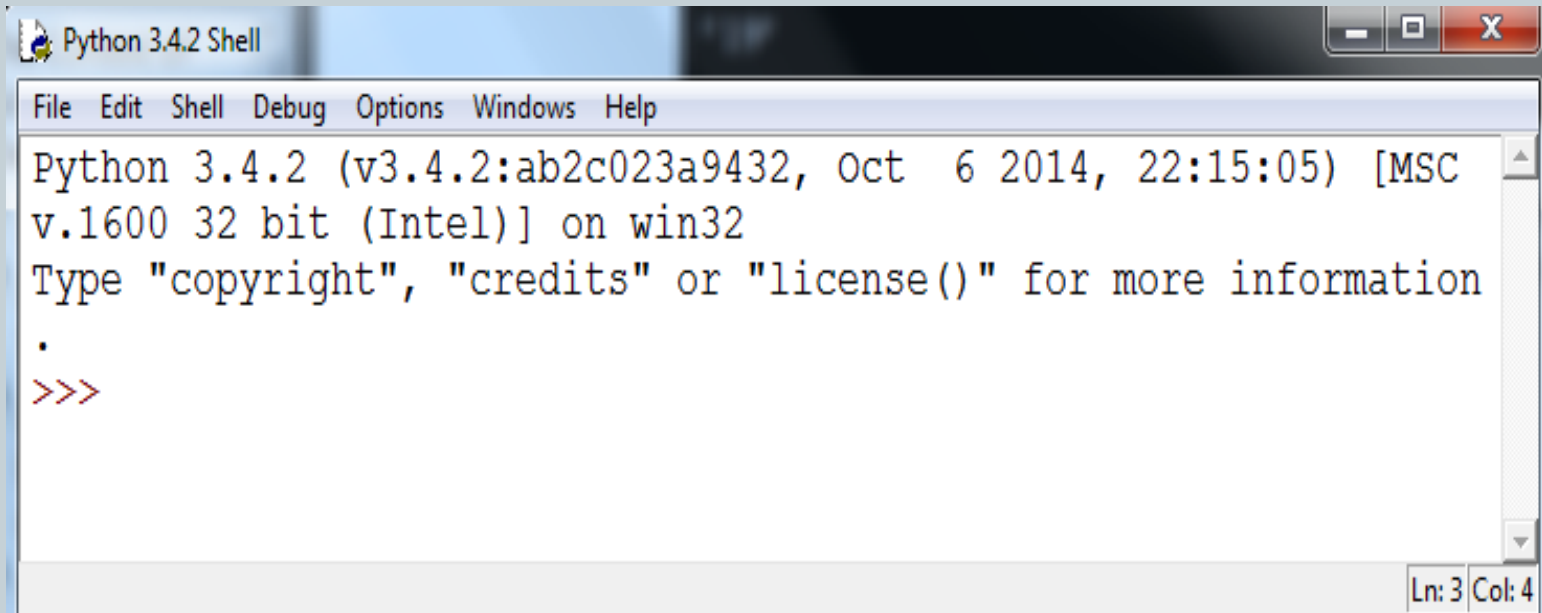
- Python has relatively few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language in a relatively short period of time.
- You can download the latest version by visiting the Python home page, at <http://www.python.org>.
- Python has an interpreter.
- Unlike human languages, the Python vocabulary is actually pretty small.
- The biggest pitfall with programming in C or C++ is that the responsibility of memory management is in the hands of the developer. The memory management is performed by the Python interpreter.

# IDLE



- An IDE is an integrated development environment—one program that provides all the tools that developer needs to write software.
- To exit the Python interactive prompt, we need to use an end-of-file character. Under Windows, this corresponds to the Ctrl-Z key combination; in Linux, it corresponds to Ctrl-D.
- Alternatively, one can use the `exit()` function:

# IDLE (Cont)

A screenshot of a Python 3.4.2 Shell window. The window has a title bar that says "Python 3.4.2 Shell" and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Windows, and Help. The main area of the window is a text editor showing the following text:

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC  
v.1600 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information  
.  
>>>
```

The text is in a monospaced font. The prompt ">>>" is in red. At the bottom right of the window, there is a status bar that says "Ln: 3 Col: 4".

# Reserved words



## Keywords

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

# Variables



- A **variable** can be any combination of letters, digits and underscore characters.
- The first character cannot be a digit.
- Variables in Python are **case sensitive**: variable and VARIABLE are not the same.
- Reserved word cannot be used for the variables.
- It should be a meaningful name.  
WWW.64BITS.LK
- Spacing is not allowed for the variables.
- Example : X      \_LabName      RESULT2  
VaRiAbLe



# Variables (Cont)



## Executed Code:

### Variable Assignment

```
a = 3  
b = a  
c = a  
a = "hello"
```

### Variables

b

bound to

c

bound to

a

### Values in Memory

type: int  
value: 3

type: string  
value: "hello"

# Variables (Cont)



## Assigning Values to Variables:

- Python variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.

# Variables (Cont)



```
counter = 100          # An integer assignment  
miles    = 1000.0      # A floating point  
name     = "John"      # A string
```

```
a = b = c = 1
```

```
a, b, c = 1, 2, "john"
```

# Built in types of Data



- A *data type* is a set of values and a set of operations defined on those values. Many data types are built into the Python language.
- In this section, we consider Python's built-in data types `int` (for integers), `float` (for floating-point numbers), `str` (for sequences of characters) and `bool` (for true-false values).

# Built in types of Data (Cont)



<i>type</i>	<i>set of values</i>	<i>common operators</i>	<i>sample literals</i>
int	<i>integers</i>	+ - * // % **	99 12 2147483647
float	<i>floating-point numbers</i>	+ - * / **	3.14 2.5 6.022e23
bool	<i>true-false values</i>	and or not	True False
str	<i>sequences of characters</i>	+	'AB' 'Hello' '2.5'

*Basic built-in data types*

# type() function



```
>>> a = 45
>>> type(a)
<type 'int'>
>>> b = 'This is a string'
>>> type(b)
<type 'str'>
>>> c = 2 + 1j
>>> type(c)
<type 'complex'>
>>> d = [1, 3, 56]
>>> type(d)
<type 'list'>
```

# Mathematics Operators



<i>Name</i>	<i>Meaning</i>	<i>Example</i>	<i>Result</i>
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

# Mathematics Operators (Cont)



Expression	Meaning
$x + y$	$x$ added to $y$ , if $x$ and $y$ are numbers $x$ concatenated to $y$ , if $x$ and $y$ are strings
$x - y$	$x$ take away $y$ , if $x$ and $y$ are numbers
$x * y$	$x$ times $y$ , if $x$ and $y$ are numbers $x$ concatenated with itself $y$ times, if $x$ is a string and $y$ is an integer $y$ concatenated with itself $x$ times, if $y$ is a string and $x$ is an integer
$x / y$	$x$ divided by $y$ , if $x$ and $y$ are numbers
$x // y$	Floor of $x$ divided by $y$ , if $x$ and $y$ are numbers
$x \% y$	Remainder of $x$ divided by $y$ , if $x$ and $y$ are numbers
$x ** y$	$x$ raised to $y$ power, if $x$ and $y$ are numbers



# Mathematics Operators (Cont)



<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Equivalent</i>
<code>+=</code>	Addition assignment	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Subtraction assignment	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Multiplication assignment	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Float division assignment	<code>i /= 8</code>	<code>i = i / 8</code>
<code>//=</code>	Integer division assignment	<code>i //= 8</code>	<code>i = i // 8</code>
<code>%=</code>	Remainder assignment	<code>i %= 8</code>	<code>i = i % 8</code>
<code>**=</code>	Exponent assignment	<code>i **= 8</code>	<code>i = i ** 8</code>

# Type the commands and get the output



```
>>> 3+4
```

```
7
```

```
>>> 3-4
```

```
-1
```

```
>>> 3/4
```

```
0.75
```

```
>>> 8/2
```

```
4.0
```

```
>>> 7%3
```

```
1
```

```
>>> -3/4
```

```
-0.75
```

```
>>> 5*4
```

```
20
```

```
>>> 3/-4
```

```
-0.75
```

```
>>> -8/2
```

```
-4.0
```

```
>>> -8/2.0
```

```
-4.0
```

```
>>> 2**4
```

```
16
```

```
>>> -2**4
```

```
-16
```

```
>>> (-2)**4
```

```
16
```

```
>>> 2**-4
```

```
0.0625
```

```
>>> 7//2
```

```
3
```

```
>>> 7//2.0
```

```
3.0
```

```
>>> -7//2
```

```
-4
```

```
>>> 7//-2
```

```
-4
```

```
>>> -7//-2
```

```
3
```

```
>>> 7%2
```

```
1
```

```
>>> -7%2
```

```
1
```

```
>>> 7%-2
```

```
-1
```

# Type the following commands and explain the output

- `-20// 3`

- `-20%3`

# Operator Precedence



## Operator Precedence

**	Exponentiation
+x, -x, ~x	Positive, negative, bitwise NOT
*, /, //, %	Multiplication, division, remainder
+, -	Addition and subtraction
<<, >>	Shifts
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
not	Boolean NOT
and	Boolean AND
or	Boolean OR

# Print() function



- A program consists of one or more statements. A statement is an instruction that the interpreter executes.
- The following statement invokes the print function to display a message:
- `print("This is a simple Python program")`
- By default, the print function places a single space in between the items it prints. Print uses a keyword argument named `sep` to specify the string to use insert between items. The name `sep` stands for separator.

# Print() function (Cont)

```
w, x, y, z = 10, 15, 20, 25
print(w, x, y, z)
print(w, x, y, z, sep=',')
print(w, x, y, z, sep='')
print(w, x, y, z, sep=':')
print(w, x, y, z, sep='-----')
```

```
10 15 20 25
10,15,20,25
10152025
10:15:20:25
10-----15-----20-----25
```

The expression `end=""` is known as a keyword argument.

```
print('A', end='')
print('B', end='')
print('C', end='')
print()
print('X')
print('Y')
print('Z')
```

```
ABC
X
Y
Z
```

# Comments



- In a Python command, anything after a # symbol is a comment.
- Comments are not part of the command, but rather intended as documentation for anyone reading the code. Multiline comments are also possible, and are enclosed by triple double-quote symbols:
- For example:  

```
print( " Hello world ") # this is a comment
```

# input() function



- The built-in function `input()` takes a string argument. This string is used as a prompt. When the `input()` function is called, the prompt is printed and the program waits for the user to provide input via the keyboard.
- `input()`: Prompts the user for input with its string argument and returns the string the user enters.



# Find the difference between the outputs:



```
a=input('Enter a number:')  
print(a)
```

```
a=int(input('Enter a number:'))  
print(a)
```

```
a=float(input('Enter a number:'))  
print(a)
```

# Built-in Functions



- `int()`: Returns the integer form of its argument.
- `float()`: Returns the float form of its argument.
- `str()`: Returns the string of its argument.
- `Bool()`: Returns the Boolean value True or False
- `chr()`: Returns the character for the ASCII value
- `ord()`: Returns the ASCII value
- `eval()`: Returns the result of evaluating its string argument as any Python expression, including arithmetic and numerical expressions

# Exercise



- Write a program to read character and get the corresponding ASCII value for the character entered.
- Write a program to read a number and get the corresponding character from the ASCII table.
- Note: Open a new file in file menu and save the file with the extension of .py

# Built-in Functions (Cont)



```
>>> int('20')
20
>>> int(2.02)
2
>>> int(True)
1
>>> int(False)
0
>>> float('20.2')
20.2
```

```
>>> str(20)
'20'
>>> bool(2)
True
>>> bool(0)
False
>>> bool(-2)
True
>>> eval('3.9')
3.9
>>> chr(65)
'A'
>>> ord('A')
65
```

# Strings



- A string is created by enclosing text in quotes. You can use either single quotes, `'`, or double quotes, `"`. A triple-quote can be used for multi-line strings.
- To get the length of a string (how many characters it has), use the built-in function **len**. For example, **len('Hello')** is 5

# Strings Concatenation and repetition



- The operators + and \* can be used on strings.
- The + operator combines two strings. This operation is called concatenation.
- The \* repeats a string a certain number of times.

# Strings Concatenation and repetition (Cont)



Expression	Result
<code>'AB'+'cd'</code>	<code>'ABcd'</code>
<code>'A'+'7'+'B'</code>	<code>'A7B'</code>
<code>'Hi '*4</code>	<code>'HiHiHiHi'</code>

# Indexing



- We will often want to pick out individual characters from a string.
- Python uses square brackets to do this.
- The table below gives some examples of indexing the string
- `s='Python'`

Statement	Result	Description
<code>s[0]</code>	P	first character of <code>s</code>
<code>s[1]</code>	y	second character of <code>s</code>
<code>s[-1]</code>	n	last character of <code>s</code>
<code>s[-2]</code>	o	second-to-last character of <code>s</code>



# Slices



- A slice is used to pick out part of a string. It behaves like a combination of indexing and the **range** function.
- Below we have some examples with the string
- `s='abcdefghij'`.

```
index:      0 1 2 3 4 5 6 7 8 9
letters:    a b c d e f g h i j
```

Code	Result	Description
<code>s[2:5]</code>	cde	characters at indices 2, 3, 4
<code>s[:5]</code>	abcde	first five characters
<code>s[5:]</code>	fghij	characters from index 5 to the end
<code>s[-2:]</code>	ij	last two characters
<code>s[:]</code>	abcdefghij	entire string
<code>s[1:7:2]</code>	bdf	characters from index 1 to 6, by twos
<code>s[: :-1]</code>	jihgfedcba	a negative step reverses the string

# Write the code and get the outputs?



- `S='ABCDEFGHIJKL'`
- 1. `S[2:6]`
- 2. `S[:8]`
- 3. `S[3:]`
- 4. `S[2:9:2]`
- 5. `S[-3:]`
- 6. `S[:]`
- 7. `S[::-1]`
- 8. `len(S)`

# The range function



Statement	Values generated
<code>range(10)</code>	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
<code>range(1, 10)</code>	1, 2, 3, 4, 5, 6, 7, 8, 9
<code>range(3, 7)</code>	3, 4, 5, 6
<code>range(2, 15, 3)</code>	2, 5, 8, 11, 14
<code>range(9, 2, -1)</code>	9, 8, 7, 6, 5, 4, 3

# The range function (Cont)



- `range(10)` → 0,1,2,3,4,5,6,7,8,9
- `range(1, 10)` → 1,2,3,4,5,6,7,8,9
- `range(1, 10, 2)` → 1,3,5,7,9
- `range(10, 0, -1)` → 10,9,8,7,6,5,4,3,2,1
- `range(10, 0, -2)` → 10,8,6,4,2
- `range(2, 11, 2)` → 2,4,6,8,10
- `range(-5, 5)` → -5,-4,-3,-2,-1,0,1,2,3,4
- `range(1, 2)` → 1
- `range(1, 1)` → (empty)
- `range(1, -1)` → (empty)
- `range(1, -1, -1)` → 1,0

# Execute the following codes:



```
for i in range(10):  
    print(i)
```

```
for i in range(10):  
    print(i, end=' ')
```

```
for i in range(3,9):  
    print(i)
```

# Conditional Statement



```
a=20
if (a==10):
    print('a is equal to 10')
    print(' a is an integer')
else:
    print('a is not equal to 10')
```

```
if x1 >= x2:
    if x1 >= x3:
        max = x1
    else:
        max = x3
else:
    if x2 >= x3:
        max = x2
    else:
        max = x3
```

# Comparison Operators



- The comparison operators are ==, >, <, >=, <=, and !=
- That last one is for not equals.
- Here are a few examples

Expression	Description
<code>if x &gt; 3:</code>	if x is greater than 3
<code>if x &gt;= 3:</code>	if x is greater than or equal to 3
<code>if x == 3:</code>	if x is 3
<code>if x != 3:</code>	if x is not 3

# Exercise



1. Write a python program to read the two numbers and find the largest number.
2. Write a python program to read the three numbers and find the minimum number.
3. Write a python program to read a number and display the number as odd or even.



# Iteration (Loops)



- Computers are often used to automate repetitive tasks.

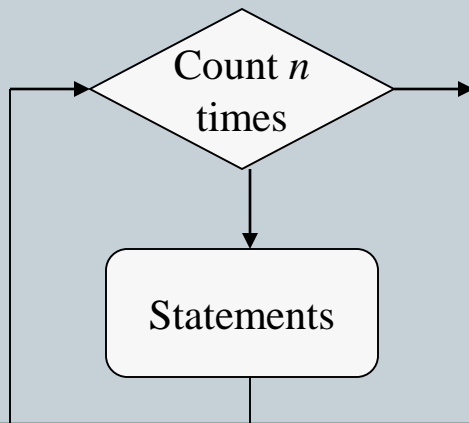
## **For loops**

- Probably the most powerful thing about computers is that they can repeat things over and over very quickly.

# Iteration and Loops



- A *loop* repeats a sequence of statements
- A *definite loop* repeats a sequence of statements a predictable number of times



```
for count in range(5): print('Hello!')
```

```
Hello  
Hello  
Hello  
Hello  
Hello
```

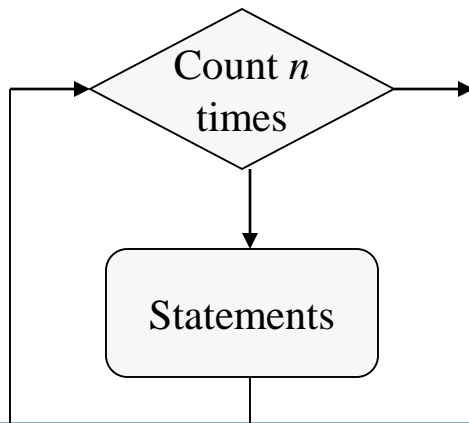
# The **for** Loop



Python's **for** loop can be used to iterate a definite number of times

```
for <variable> in range(<number of times>): <statement>
```

Use this syntax when you have only one statement to repeat



```
for count in range(5): print('Hello!')
```

```
Hello  
Hello  
Hello  
Hello  
Hello
```

# The for Loop

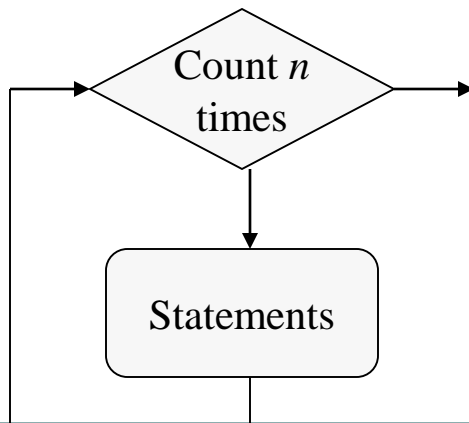


```
for <variable> in range(<number of times>):  
    <statement-1>  
    <statement-2>  
    ...  
    <statement-n>
```

← loop header

← loop body

Use *indentation* to format two or more statements below the *loop header*



```
for count in range(3):  
    print('Hello!')  
    print('goodbye')
```

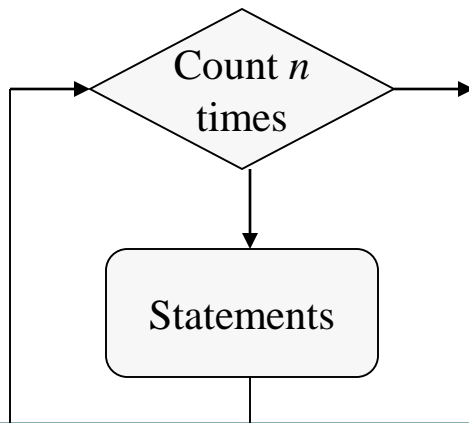
```
Hello!  
goodbye  
Hello!  
goodbye  
Hello!  
goodbye
```

# Using the Loop Variable



The *loop variable* picks up the next value in a *sequence* on each pass through the loop

The expression **range(n)** generates a sequence of **ints** from **0** through **n - 1**



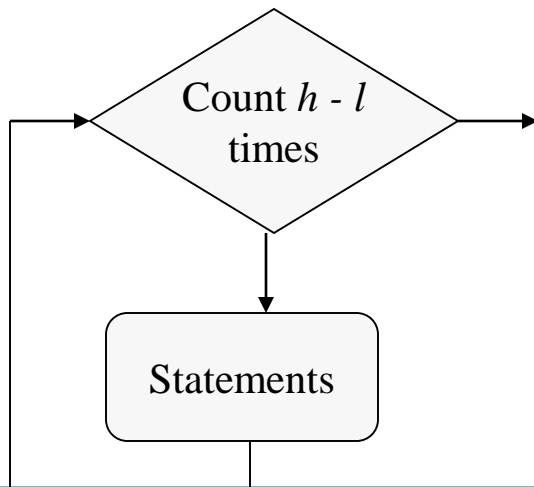
loop variable

```
>>> for count in range(5):  
    print(count)  
...  
0  
1  
2  
3  
4  
>>> list(range(5)) # Show as a list  
[0, 1, 2, 3, 4]
```

# Counting from 1 through $n$



The expression `range(low, high)` generates a sequence of `ints` from `low` through `high - 1`

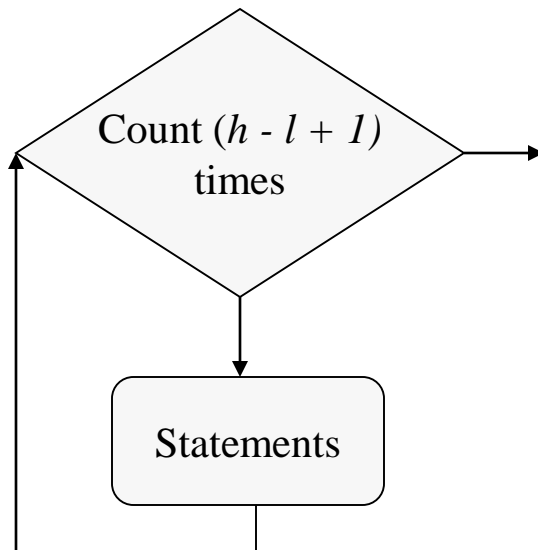


```
>>> for count in range(1, 6): print(count)
...
1
2
3
4
5
>>> list(range(1, 6)) # Show as a list
[1, 2, 3, 4, 5]
```

# Skipping Steps in a Sequence



The expression **range(low, high, step)** generates a sequence of **ints** starting with **low** and counting by **step** until **high - 1** is reached or exceeded

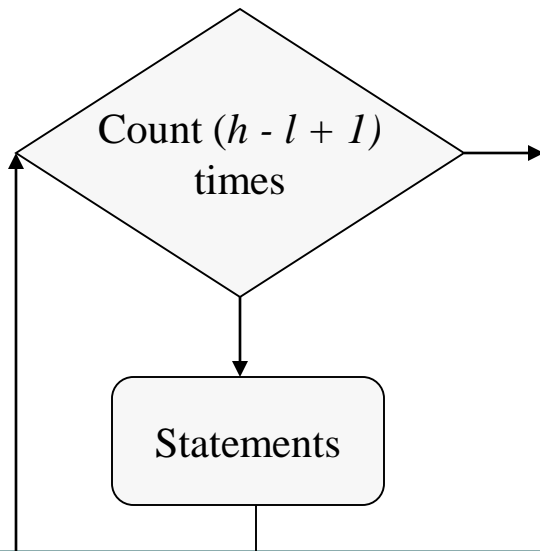


```
>>> for count in range(1, 6, 2):  
    print(count)  
...  
1  
3  
5  
>>> list(range(1, 6, 2)) # Show as a list  
[1, 3, 5]
```

# Counting down in a Sequence



The expression **range(high, low, step)** generates a sequence of **ints** starting with **high** and counting by **step** until **low - 1** is reached, when **step** is negative



```
>>> for count in range(4, 1, -1):  
    print(count)  
...  
4  
3  
2  
>>> list(range(4, 1, -1)) # Show as a list  
[4, 3, 2]
```



# Accumulator Loop: Summation



Compute and print the sum of the numbers between 1 and 5, inclusive

```
total = 0
for n in range(1, 6):
    total = total + n
print(total)
```

In this program, the variable **total** is called an *accumulator variable*

# Iteration (Cont)



```
for i in range(3):  
    num = eval(input('Enter a number: '))  
    print ('The square of your number is', num*num)  
print('The loop is now done.')
```

Output

```
Enter a number: 3  
The square of your number is 9  
Enter a number: 5  
The square of your number is 25  
Enter a number: 23  
The square of your number is 529  
The loop is now done.
```

# Iteration (Cont)



```
>>> for i in range(7): # Header for outer loop.  
...     for j in range(1, i + 2): # Cycle through integers.  
...         print(j, end="")      # Suppress newline.  
...     print()                  # Add newline.
```

...

1

12

123

1234

12345

123456

1234567

Output

# Iteration (Cont)



## While Loop

```
while (conditional test):  
    Statement 01  
    Statement 02  
    .....  
    Statement n
```

- While something is True keep running the loop, exit as soon as the test is False.
- The conditional test syntax is the same as for if and elif statements.

# Iteration (Cont)



```
for i in range(10):  
    print(i)
```

```
i=0  
while i<10:  
    print(i)  
    i=i+1
```

# The break statement



- The **break** statement can be used to break out of a for or while loop before the loop is finished

```
for i in range(10):  
    num = eval(input('Enter number: '))  
    if num < 0:  
        break
```

# The Continue statement



- This is a control flow statement that causes the program to immediately skip the processing of the rest of the body of the loop, *for the current iteration*.
- But the loop still carries on running for its remaining iterations:

```
for i in [12, 16, 17, 24, 29, 30]:  
    if i % 2 == 1:        # if the number is odd  
        continue        # don't process it  
    print(i)  
print("done")
```

# The pass Statement



- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.



# Exercise



Exercise: Find the output of the following Python code:

```
>>> a=1
```

```
>>> while (a<=5) :
```

```
    print (a)
```

```
    a=a+1
```

```
>>> c=1
```

```
>>> while (c<=5) :
```

```
    print (c, end=' ')
```

```
    c=c+1
```

```
>>> b=1
```

```
>>> while (b<=5) :
```

```
    print (b, end="")
```

```
    b=b+1
```

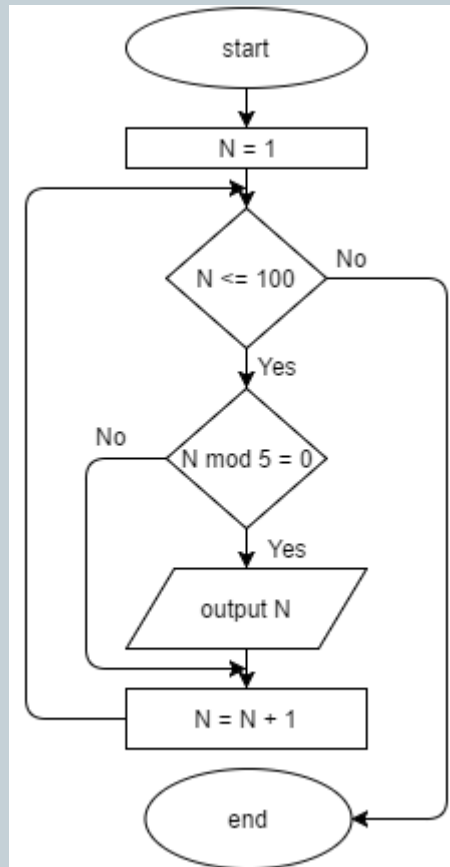
```
>>> c=1
```

```
>>> while (c<=5) :
```

```
    print (c, end="#")
```

```
    c+=1
```

# Implement the following flowchart in Python and get the output.





**QUESTIONS??**



**THANK YOU!!**