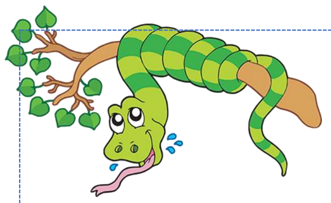


Python Flow Control

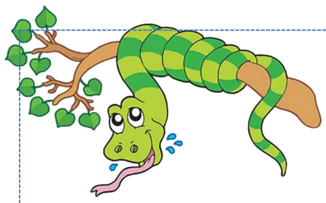
(Flowcharts, Boolean Values, Comparison Operators)

Class viii

Lab 19

UNITED TRUST
ADD STRENGTHS TO THE WEAK

Powered By
bdeducation
Bangladesh Digital Education Research Limited
সময়ের পাঠশালা



Lab Objectives:

- Flowchart
- If, else & elif statements
- Flow Control Statements
- Turn Flowchart statements into python code

So you know the basics of individual instructions and that a program is just a series of instructions. But the real strength of programming isn't just running (or executing) one instruction after another like a weekend To Do list. Based on how the expressions evaluate, the program can decide to skip instructions, repeat them, or choose one of several instructions to run. In fact, you almost never want your programs to start from the first line of code and simply execute every line, straight to the end. Flow control statements can decide which Python instructions to execute under which conditions.

Flowchart

These flow control statements directly correspond to the symbols in a flowchart, so we'll provide flowchart versions of the code discussed in this chapter. Figure 1 shows a flowchart for what to do if it's raining. Follow the path made by the arrows from Start to End.

Your target is to run from start to end. If rain is happening there is one way if not happening, there's another way. Besides you may have umbrella or not. If you have you can go through rain otherwise, you have to stay at home.

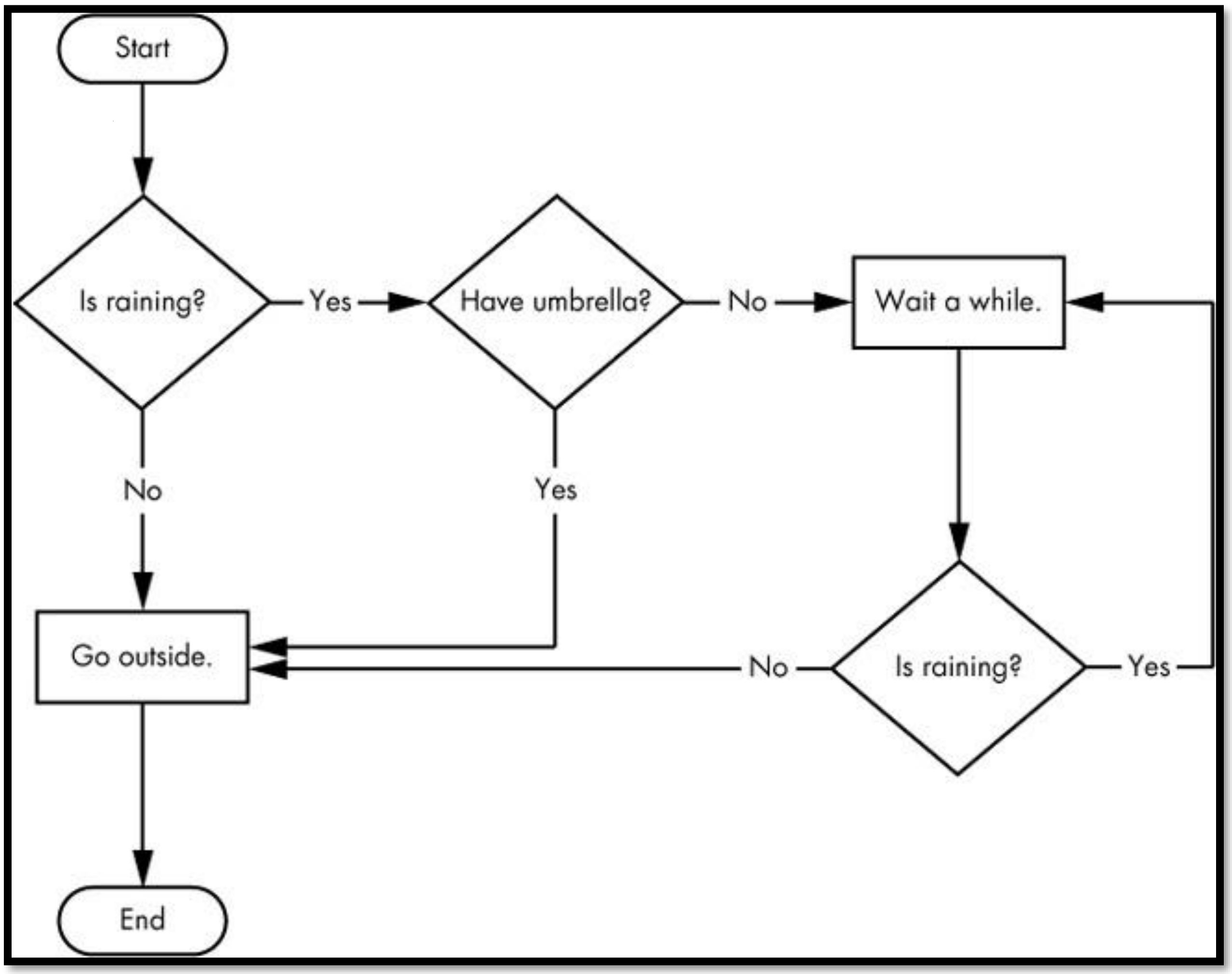
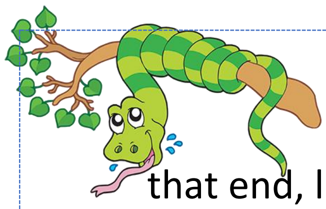


Figure 1

In a flowchart, there is usually more than one way to go from the start to the end. The same is true for lines of code in a computer program. Flowcharts represent these branching points with diamonds, while the other steps are represented with rectangles. The starting and ending steps are represented with rounded rectangles.

But before you learn about flow control statements, you first need to learn how to represent those yes and no options, and you need to understand how to write those branching points as Python code. To



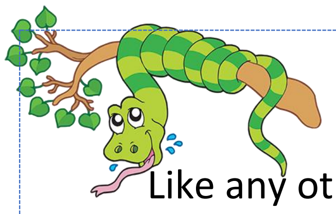
that end, let’s explore Boolean values, comparison operators, and Boolean operators.

Boolean Values

While the integer, floating-point, and string data types have an unlimited number of possible values, the Boolean data type has only two values: True and False. (Boolean is capitalized because the data type is named after mathematician George Boole.) When typed as Python code, the Boolean Values True and False lack the quotes you place around strings, and they always start with a capital T or F, with the rest of the word in lowercase.

Enter the following into the interactive shell. (Some of these instructions are intentionally incorrect, and they’ll cause error messages to appear.)

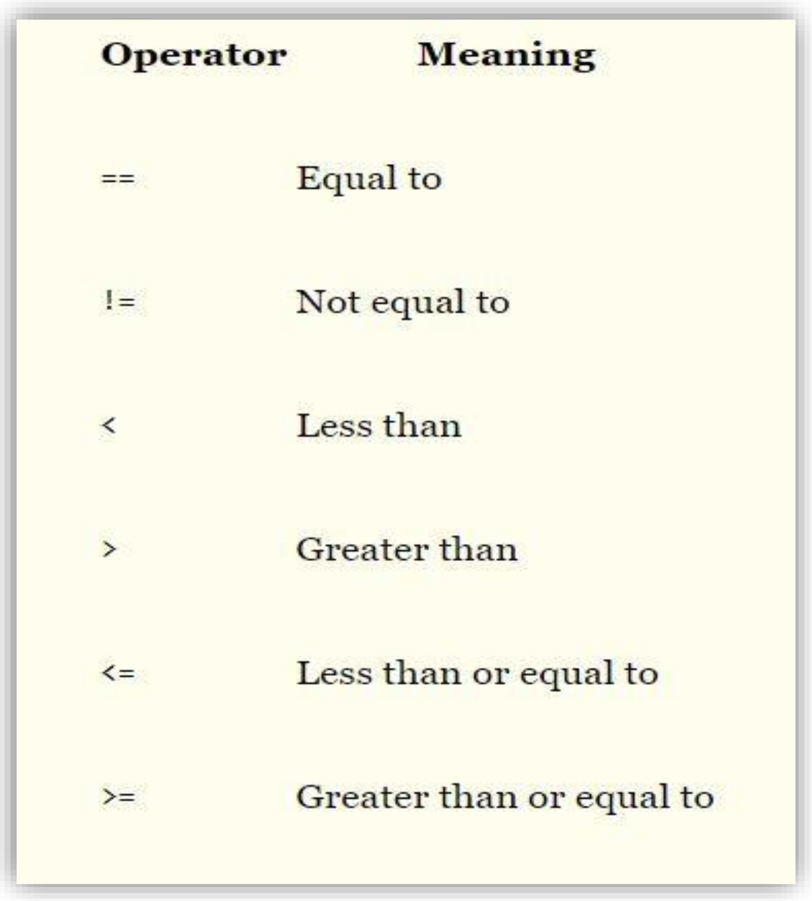
```
>>> spam = True

>>> spam

True


>>> true

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

NameError: name 'true' is not defined


>>> True = 2 + 2

  File "<stdin>", line 1

SyntaxError: can't assign to keyword
```



Like any other value, Boolean values are used in expressions and can be stored in variables ❶. If you don't use the proper case ❷ or you try to use True and False for variable names ❸, Python will give you an error message.

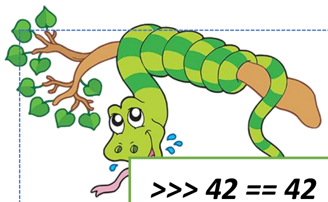
Comparison Operators

Comparison operators compare two values and evaluate down to a single Boolean value. Table 2 lists the comparison operators.



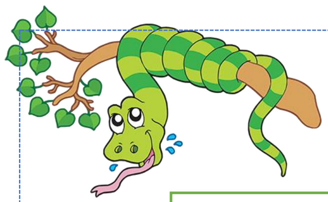
Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

These operators evaluate to True or False depending on the values you give them. Let's try some operators now, starting with == and !=.



```
>>> 42 == 42
True


>>> 42 == 99
False



>>> 2 != 3
True
>>> 2 != 2
False
>>>
```

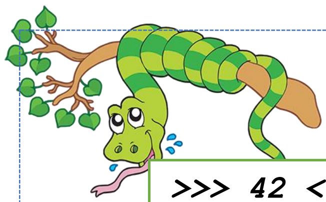
As you might expect, == (equal to) evaluates to True when the values on both sides are the same, and != (not equal to) evaluates to True when the two values are different. The == and != operators can actually work with values of any data type.



```
>>> 'hello' == 'hello'

True


>>> 'hello' == 'Hello'

False


>>> 'dog' != 'cat'

True


>>> True == True

True


>>> True != False

True
```

Note that an integer or floating-point value will always be unequal to a string value. The expression 42 == '42' ❶ evaluates to False because Python considers the integer 42 to be different from the string '42'.

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

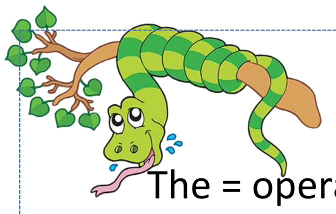


```
>>> 42 < 100
True

>>> 42 > 100
False

>>> 42 < 42
False

>>> eggCount = 42
>>> eggCount <= 42
True

>>> myAge = 29
>>> myAge >= 10
True
>>>
```

The Difference Between the == and = Operators:

You might have noticed that the == operator (equal to) has two equal signs, while the = operator (assignment) has just one equal sign. It’s easy to confuse these two operators with each other. Just remember these points:

The == operator (equal to) asks whether two values are the same as each other.



The = operator (assignment) puts the value on the right into the variable on the left.

To help remember which is which, notice that the == operator (equal to) consists of two characters, just like the != operator (not equal to) consists of two characters.

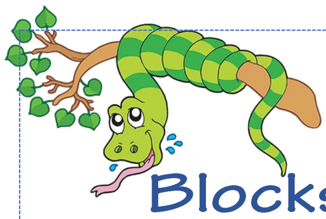
You'll often use comparison operators to compare a variable's value to some other value, like in the eggCount <= 42 ❶ and myAge >= 10 ❷ examples. (After all, instead of typing 'dog' != 'cat' in your code, you could have just typed True.) You'll see more examples of this later when you learn about flow control statements.

Elements of Flow Control

Flow control statements often start with a part called the condition, and all are followed by a block of code called the clause. Before you learn about Python's specific flow control statements, we'll cover what a condition and a block are.

Conditions

The Boolean expressions you've seen so far could all be considered conditions, which are the same thing as expressions; condition is just a more specific name in the context of flow control statements. 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

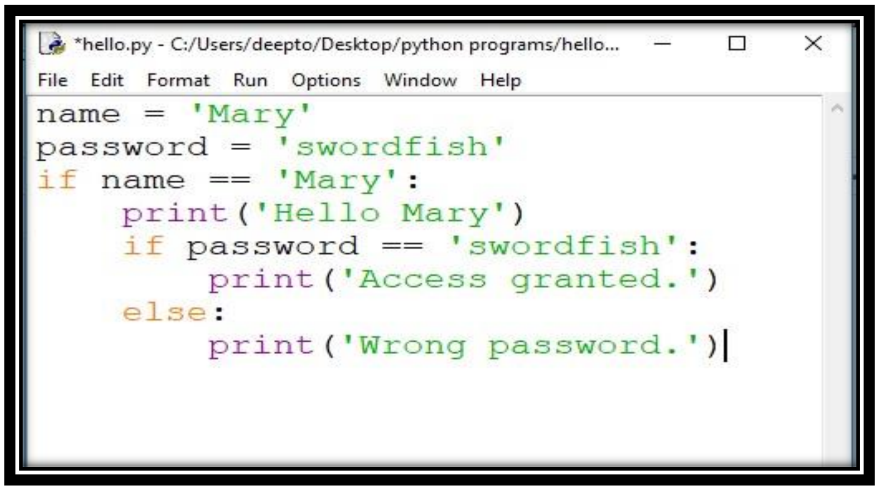
Lines of Python code can be grouped together in blocks. You can tell when a block begins and ends from the indentation of the lines of code. There are three rules for blocks.

Blocks begin when the indentation increases.

Blocks can contain other blocks.

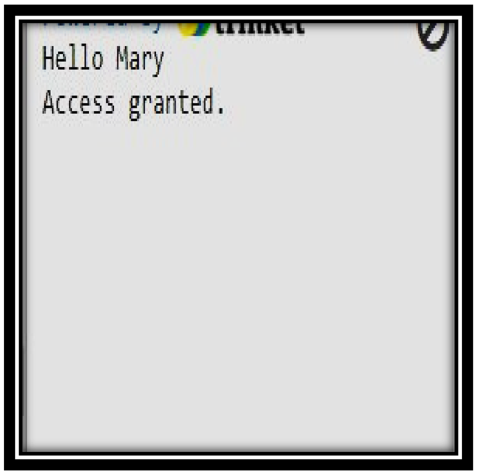
Blocks end when the indentation decreases to zero or to a containing block's indentation.

Blocks are easier to understand by looking at some indented code, so let's find the blocks in part of a small game program, shown here:



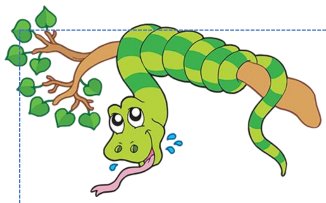
*hello.py - C:/Users/deepto/Desktop/python programs/hello...
File Edit Format Run Options Window Help

```python
name = 'Mary'
password = 'swordfish'
if name == 'Mary':
    print('Hello Mary')
    if password == 'swordfish':
        print('Access granted.')
    else:
        print('Wrong password.')
```



```
Hello Mary
Access granted.
```

The first block of code ❶ starts at the line print('Hello Mary') and contains all the lines after it. Inside this block is another block ❷, which has only a single line in it: print('Access Granted.'). The third block ❸ is also one line long: print('Wrong password.').



Flow Control Statements

Now, let’s explore the most important piece of flow control: the statements themselves.

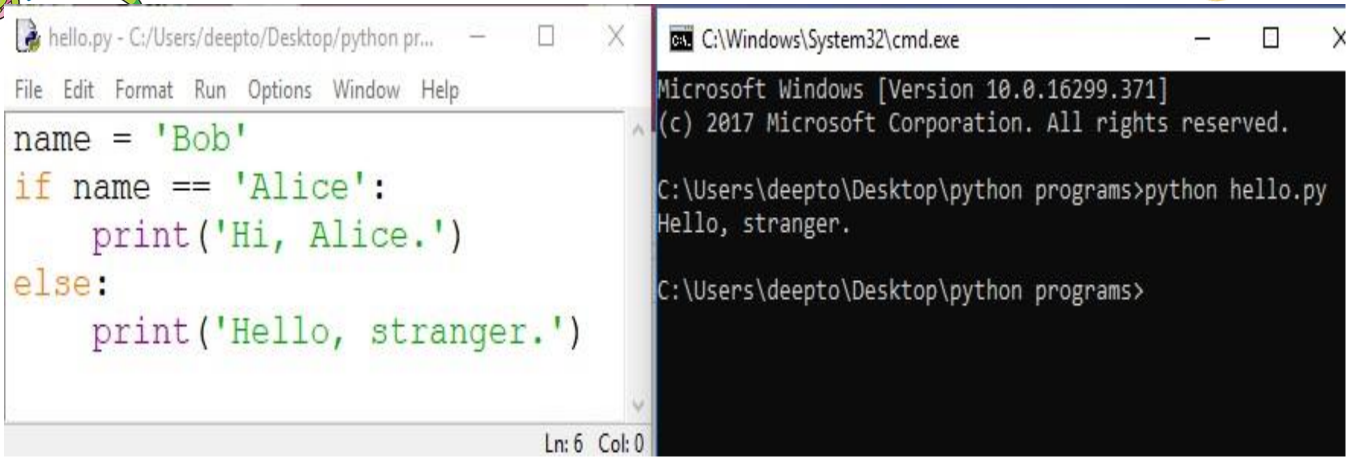
IF and Else statements

he most common type of flow control statement is the if statement. An if statement’s clause (that is, the block following the if statement) will execute if the statement’s condition is True. The clause is skipped if the condition is False.

In plain English, an if statement could be read as, “If this condition is true, execute the code in the clause.”

For example, let’s say you have some code that checks to see whether someone’s name is Alice. (Pretend name was assigned some value earlier.)

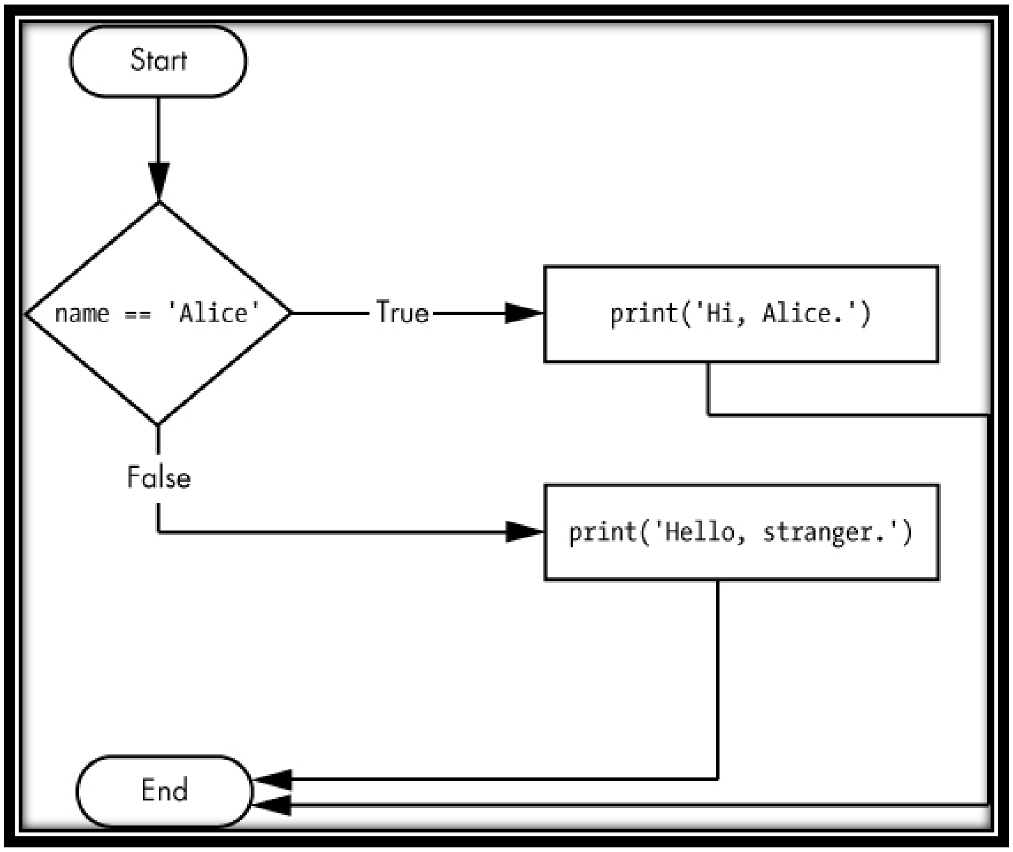
```
if name == 'Alice':
    print('Hi, Alice.')
else:
   print(“Hi, Stranger”)
```

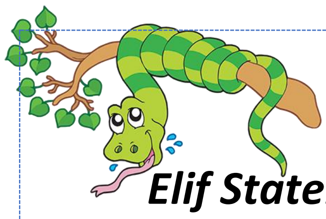

hello.py - C:/Users/deepto/Desktop/python pr...
File Edit Format Run Options Window Help
Ln: 6 Col: 0
C:\Windows\System32\cmd.exe

```python
name = 'Bob'
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\deepto\Desktop\python programs>python hello.py
Hello, stranger.

C:\Users\deepto\Desktop\python programs>
```

our coding is looking like the following flowchart





Elif Statement

While only one of the if or else clauses will execute, you may have a case where you want one of many possible clauses to execute.
The elif statement is an “else if” statement that always follows an if or another elif statement. It provides another condition that is checked only if all of the previous conditions were False.

In code, an elif statement always consists of the following:

The elif keyword

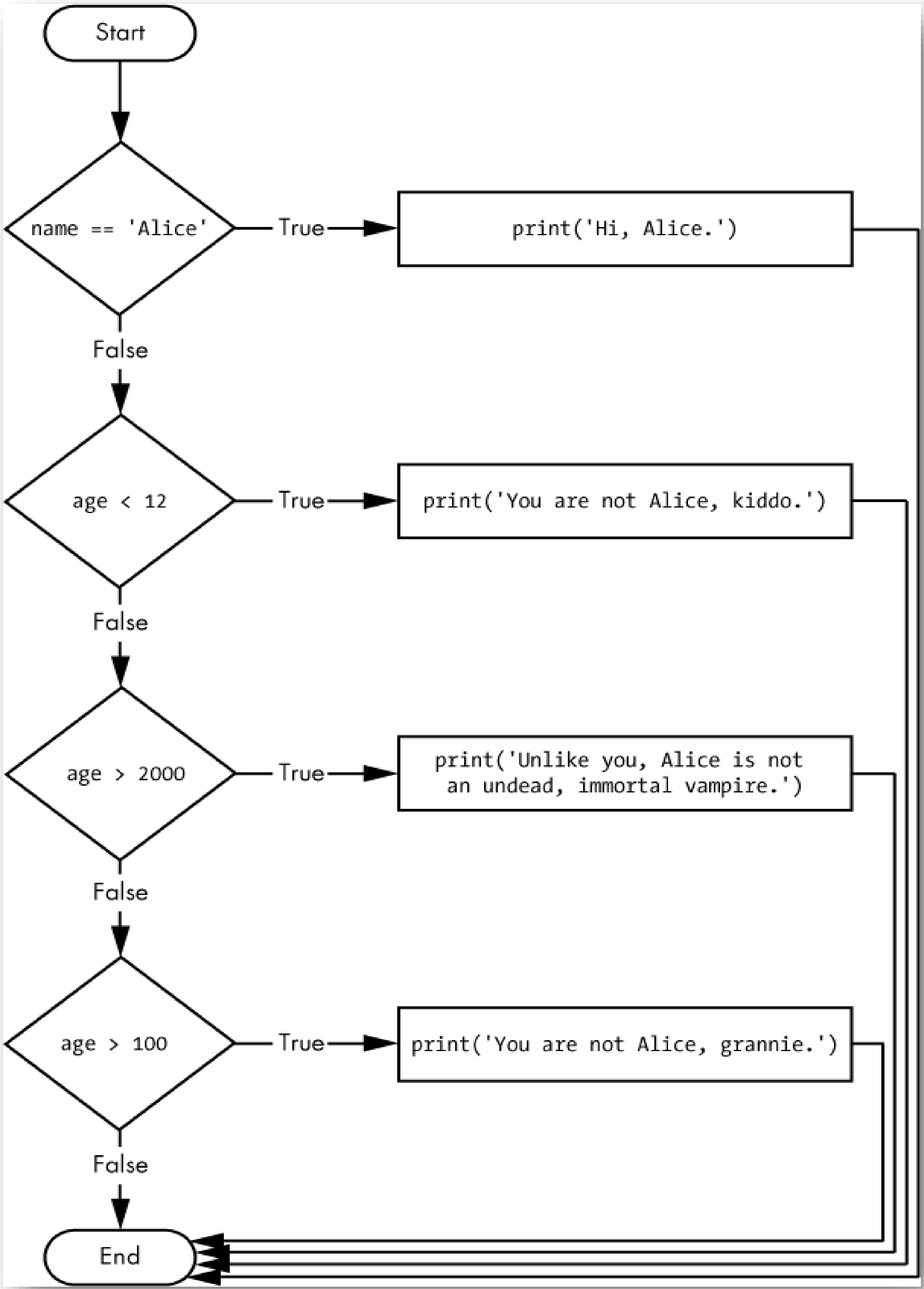
A condition (that is, an expression that evaluates to True or False)

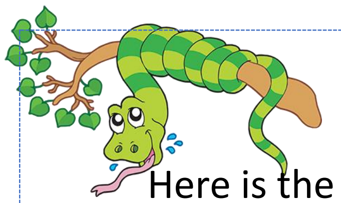
A colon

Starting on the next line, an indented block of code (called the elif clause)

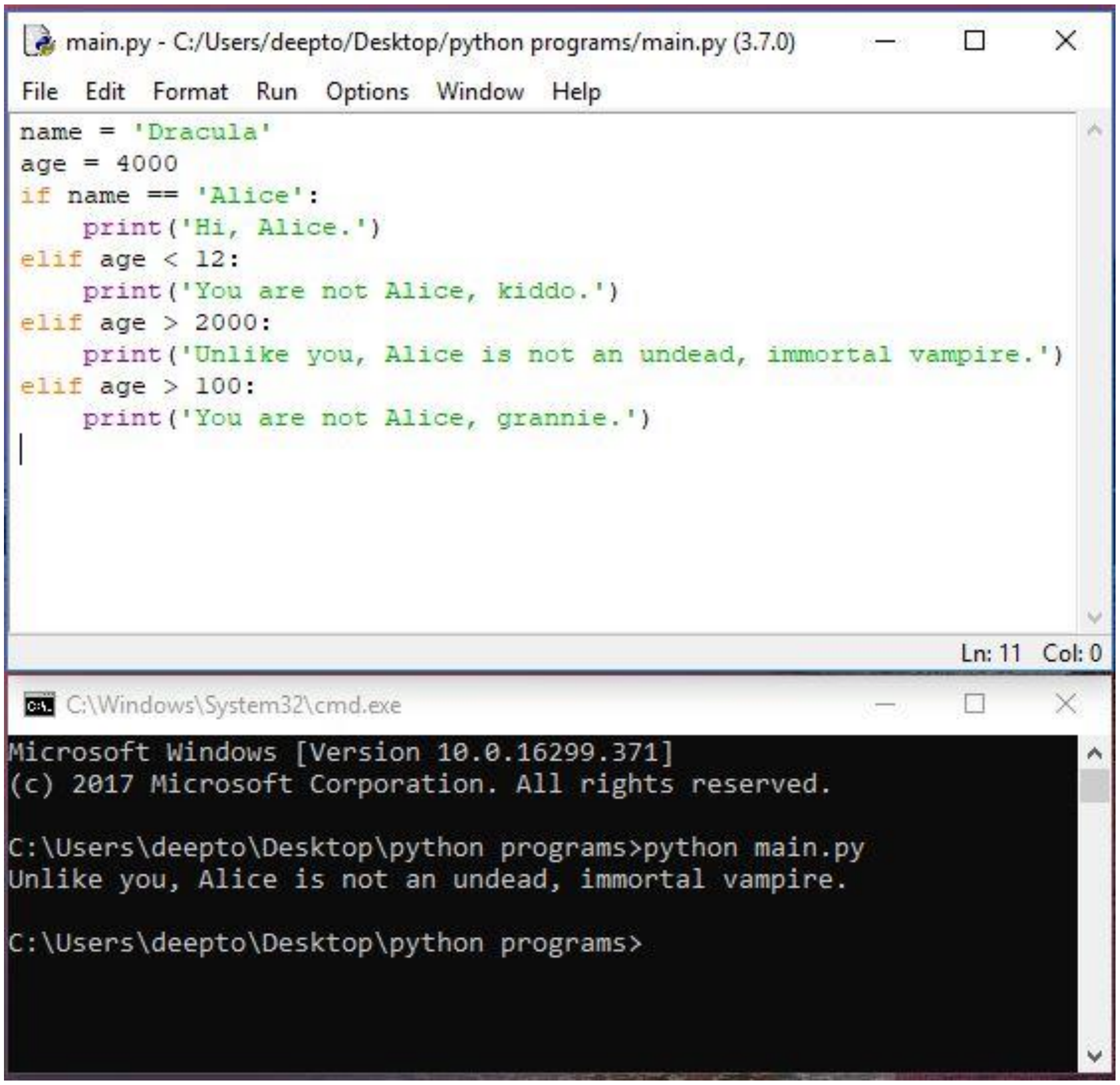
Let’s add an elif to the name checker to see this statement in action.

Follow the flowchart and try to code yourself.


Start
name == 'Alice'
True
print('Hi, Alice.')
False
age < 12
True
print('You are not Alice, kiddo.')
False
age > 2000
True
print('Unlike you, Alice is not an undead, immortal vampire.')
False
age > 100
True
print('You are not Alice, grannie.')
False
End



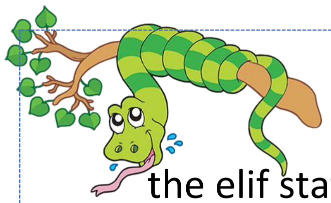
Here is the python code and its output for the following flowchart above


main.py - C:/Users/deepto/Desktop/python programs/main.py (3.7.0)
File Edit Format Run Options Window Help
Ln: 11 Col: 0
C:\Windows\System32\cmd.exe

```python
name = 'Dracula'
age = 4000
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\deepto\Desktop\python programs>python main.py
Unlike you, Alice is not an undead, immortal vampire.

C:\Users\deepto\Desktop\python programs>
```

Say the age variable contains the value 3000 before this code is executed. You might expect the code to print the string 'Unlike you, Alice is not an undead, immortal vampire.'. However, because the age > 100 condition is True (after all, 3000 is greater than 100) ❶, the string 'You are not Alice, grannie.' is printed, and the rest of



the elif statements are automatically skipped. Remember, at most only one of the clauses will be executed, and for elif statements, the order matters!