Boolean Expressions

- In Computer science, questions are often referred to as “Boolean expressions” because of the result of asking them is either True or False

Python Relational Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	c = a + b will assigne value of a + b into c

Different from

a = 10
b = 20

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.

Python Relational Operators

a = 10
b = 20

!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Logical Operators

- In Python compound Boolean expressions are composed of simple Boolean expressions that are connected by the logical operators:
  - and, or, not
- x and y - if x is False, return x; otherwise return y
- x or y - if x is False, return y; otherwise return
- not x - if x is False, return True; otherwise return False.

Exercise

- Write a compound Boolean expression that returns True if the value of the variable *count* is between 1 and 10 inclusive

The Random Module

- In order to implement our Monte Carlo simulation, we will use the *random* module

```
>>> import random
>>> help(random)
Help on module random:

NAME
    random - Random variable generators.

>>> help (random.random)
Help on built-in function random:

random(...)
    random() -> x in the interval [0, 1).
```

Selection Statements

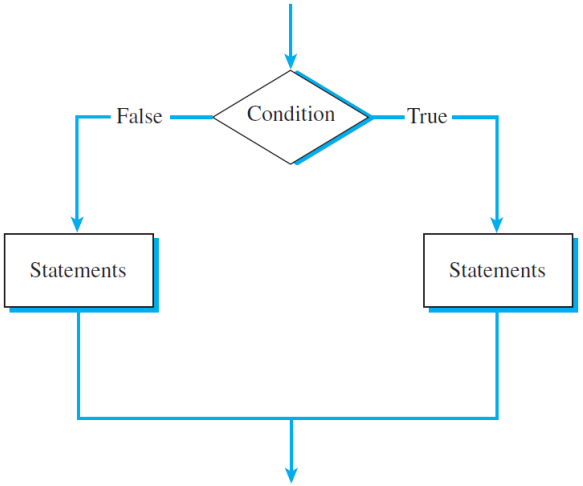
- Ask a question (Boolean Expression)
- Using that question for making decision
- Based on the answer, perform a task

The ifelse statement

if <condition>:
    <statements>
else:
    <statements>

- The keyword if is followed by a Boolean expression that is used as a condition.
- When the Boolean expression is evaluated, the result will either be True or False.
  - If it is True, the first group of statement is executed in sequence and the second group is skipped
  - If the condition is False, the first group is skipped and the second group is executed in sequence
  - Thus it is possible to perform different actions based on the result of the questions asked

The Logical Flow of an *ifelse* Statement

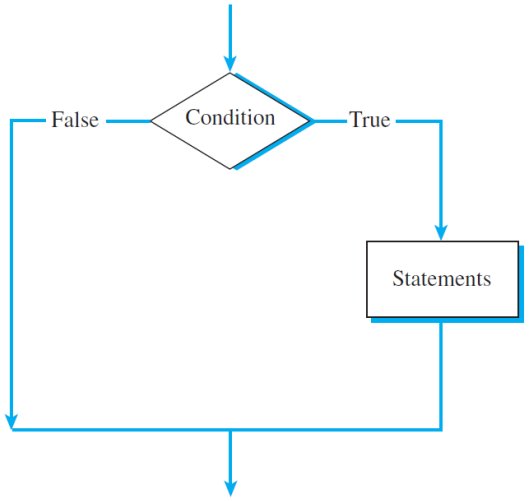


The if statement

```
if <condition>:
    <statements>
```

- As before, the condition is evaluated and the result will either be *True* or *False*.
  - If the result is True, the first group of statement is executed in sequence
  - If the condition is False, nothing is performed, the statement is complete, and the program goes on to the next statement in sequence.

The Logical Flow of an *if* Statement



The Nested *if*

```
if <condition>:
    if <condition>:
    <statements>
   else:
    <statements>
else:
    if <condition>:
    <statements>
   else:
    <statements>
```

Tail Nesting

- A common pattern for nested selection is called *tail nesting.*
  - With tail nesting, all of the nested selection occurs in the *else* part of the previous *ifelse* statement.

```
if <condition>:
    <statements>
else:
    if <condition>:
    <statements>
    else:
        if <condition>:
        <statements>
        else:
            if <condition>:
            <statements>
             …….
```

Tail Nesting

- This tail nesting pattern is so common that python provides a shorter version known as the *elif* statement

```
if <condition>:
    <statements>
else:
    if <condition>:
    <statements>
    else:
        if <condition>:
        <statements>
        else:
            if <condition>:
            <statements>
             …….
```

```
if <condition>:
    <statements>
elif <condition>:
    <statements>
elif <condition>:
    <statements>
elif <condition>:
     <statements>
    …….
else:
    <statement>
```