Programming using Python

Operators and Expressions

Binary Operations

Op	Meaning	Example	Remarks
+	Addition	9+2 is 11	
		9.1+2.0 is 11.1	
-	Subtraction	9-2 is 7	
		9.1-2.0 is 7.1	
*	Multiplication	9*2 is 18	
		9.1*2.0 is 18.2	
/	Division	9/2 is 4.5	In Python3
		9.1/2.0 is 4.55	Real div.
//	Integer Division	9//2 is 4	
%	Remainder	9%2 is 1	

The // operator

- Also referred to as “integer division”

- Result is a whole integer (floor of real division)
  - But the type need not be int
  - the integral part of the real division
  - rounded towards minus infinity ($-\infty$)

- Examples

9//4 is 2	**(-1)//2 is -1**	**(-1)//(-2) is 0**
1//2 is 0	**1//(-2) is -1**	**9//4.5 is 2.0**

The % operator

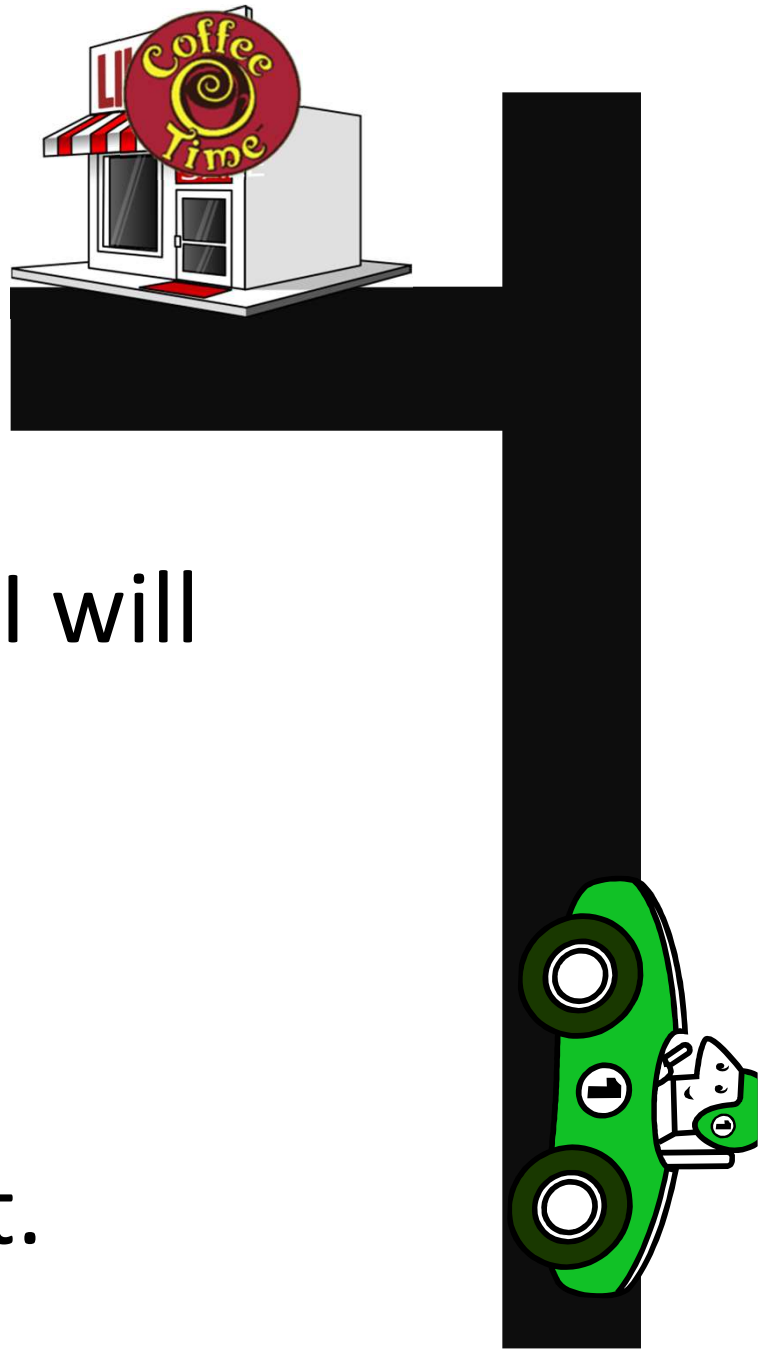
- The remainder operator % returns the remainder of the result of dividing its first operand by its second.

9%4 is 1	**(-1)%2 is 1**	**(-1)//(-2) is 0**
9%4.5 is 0.0	**1%(-2) is 1**	**1%0.6 is 0.4**

Conditional Statements

In daily routine

- If it is very hot, I will skip exercise.



- If there is a quiz tomorrow, I will first study and then sleep. Otherwise, I will sleep now.

- If I have to buy coffee, I will go left. Else I will go straight.

if-else statement

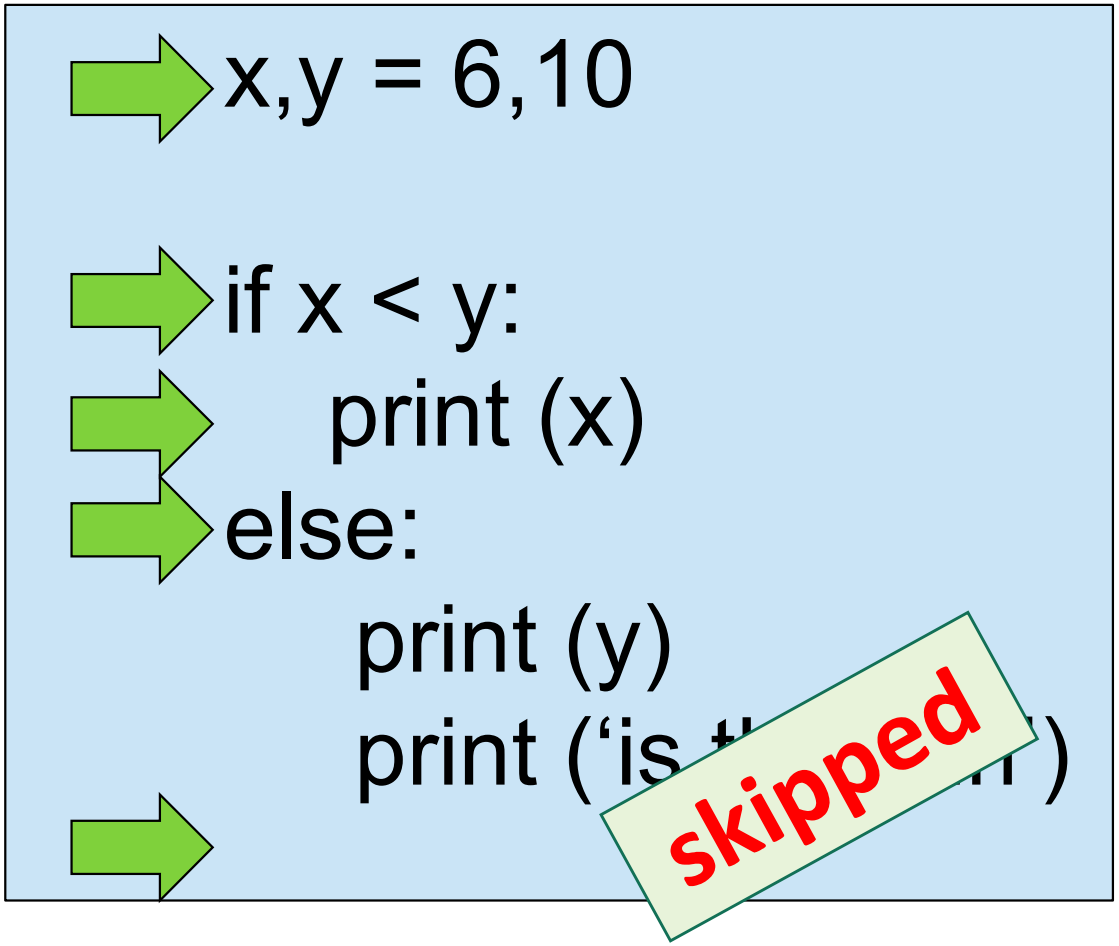
- Compare two integers and print the min.

```
if  x < y:
        print (x)
else:
        print (y)
print ('is the minimum')
```

1. Check if x is less than y.
2. If so, print x
3. Otherwise, print y.

Indentation

- Indentation is **important** in Python
  - grouping of statement (block of statements)
  - no explicit brackets, e.g. { }, to group statements



Run the program

6

10

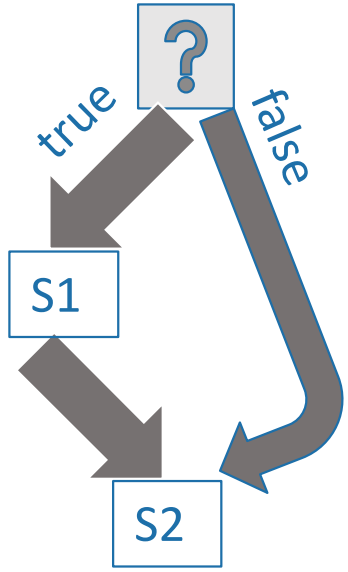
Output

6

if statement (no else!)

- General form of the if statement

```
if boolean-expr :
        S1
S2
```

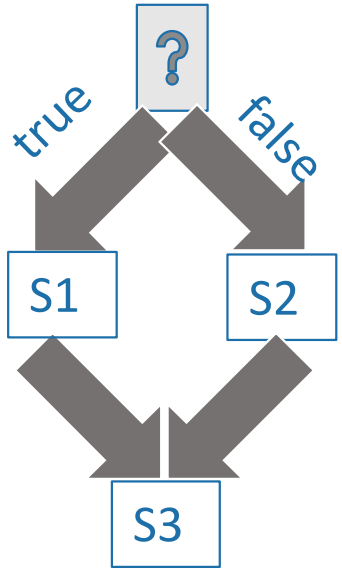


- Execution of if statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to the S2.
  - If expression evaluates to **false**, then control moves to the S2 directly.

if-else statement

- General form of the if-else statement

```
if boolean-expr :
        S1
else:
        S2
S3
```



- Execution of if-else statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to S3.
  - If expression evaluates to **false**, then S2 is executed and then control moves to S3.
  - S1/S2 can be **blocks** of statements!

Nested if, if-else

```
if a <= b:
        if a <= c:
            …
        else:
            …
else:
        if b <= c) :
            …
        else:
            …
```

Elif

- A special kind of nesting is the chain of if-else-if-else-… statements
- Can be written elegantly using if-elif-..-else

```
if cond1:
    s1
else:
    if cond2:
        s2
    else:
        if cond3:
            s3
        else:
            …
```

```
if cond1:
    s1
elif cond2:
    s2
elif cond3:
    s3
elif  …
else
    last-block-of-stmt
```

Summary of if, if-else

- if-else, nested if's, elif.
- Multiple ways to solve a problem
  - issues of readability, maintainability, and efficiency

Class Quiz

- What is the value of expression:

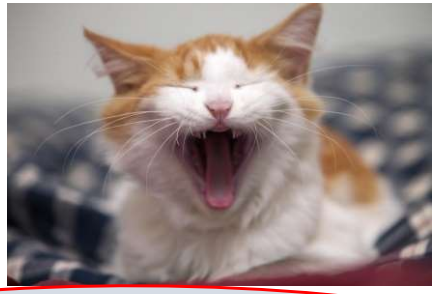
$$(5<2) \text{ and } (3/0 > 1)$$

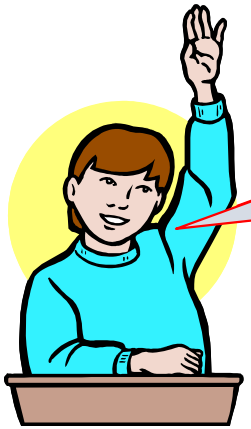
a) Run time crash/error

b) I don't know / I don't care

c) False

d) True

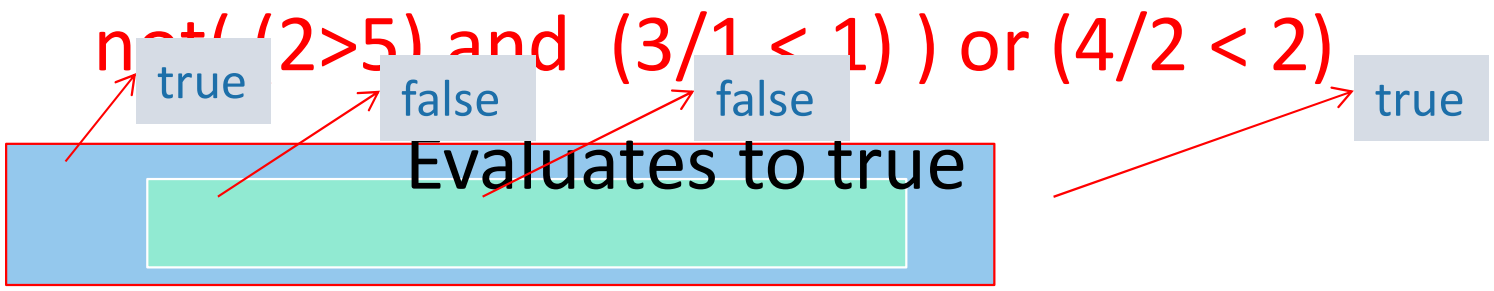




The correct answer is **a**

Short-cut Evaluation

- Do not evaluate the second operand of binary logical operator if the result can be deduced from the first operand
  - Also applies to nested logical operators



3 Factors for Expression Evaluation

- **Precedence**
  - Applied to two different operators
  - + and *, - and *, and and or, …
- **Associativity**
  - Applied to operators of same type
  - * and *, + and -, * and /, …
- **Order**
  - Precedence and associativity **identify the operands** for each operator
  - **Not which operand is evaluated first**
  - Python evaluates expressions from left to right
  - While evaluating an assignment, the right-hand side is evaluated before the left-hand side.

Class Quiz

- What is the output of the following program:

```
y = 0.1*3
if y != 0.3:
    print ('Launch a Missile')
else:
    print ("Let's have peace")
```

Launch a Missile

Caution about Using Floats

- Representation of *real numbers* in a computer can not be exact
  - Computers have limited memory to store data
  - *Between any two distinct real numbers, there are infinitely many real numbers.*
- On a typical machine running Python, there are 53 bits of precision available for a Python float

Caution about Using Floats

- The value stored internally for the decimal number 0.1 is the binary fraction

  0.0001100110011001100110011001100110011001100110011001100110011010

- Equivalent to decimal value

  0.1000000000000000055511151231257827021181583404541015625

- Approximation is similar to decimal approximation 1/3 = 0.333333333...
- No matter how many digits you use, you have an approximation

Comparing Floats

- Because of the approximations, comparison of floats is not exact.
- **Solution?**
- Instead of

$$x == y$$

use

$$abs(x-y) <= epsilon$$

where epsilon is a suitably chosen small value