# Lecture 3 - Conditionals

Adnan Ferdous Ashrafi

Stamford University Bangladesh

# Table of Contents

## Floor division

### Definition

The **floor** division operator, //, divides two numbers and rounds down to an integer. For example, suppose the run time of a movie is 105 minutes. You might want to know how long that is in hours. Conventional division returns a floating-point number:

```
>>> minutes = 105
>>> minutes / 60
1.75
```

But we don't normally write hours with decimal points. Floor division returns the integer number of hours, rounding down:

```
>>> minutes = 105
>>> hours = minutes // 60
>>> hours
1
```

## Floor division

To get the remainder, you could subtract off one hour in minutes:

```
>>> remainder = minutes - hours * 60
>>> remainder
45
```

## Modulus Operator

### Definition

The **modulus operator**, %, divides two numbers and returns the remainder.

```
>>> remainder = minutes % 60
>>> remainder
45
```

The modulus operator is more useful than it seems. For example, you can check whether one number is divisible by another—if $x\%y$ is zero, then $x$ is divisible by $y$.
Also, you can extract the right-most digit or digits from a number. For example, $x\%10$ yields the right-most digit of x (in base 10). Similarly $x\%100$ yields the last two digits.

## Boolean Expressions

### Definition

A **boolean** expression is an expression that is either true or false. The following examples use the operator ==, which compares two operands and produces *True* if they are equal and *False* otherwise:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

*True* and *False* are special values that belong to the type *bool*; they are not strings:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

## Boolean Expressions

The $==$ operator is one of the relational operators; the others are:

| | |
|---|---|
| $x! = y$ | # x is not equal to y |
| $x > y$ | # x is greater than |
| $x < y$ | # x is less than |
| $x >= y$ | # x is greater than or equal to y |
| $x <= y$ | # x is less than or equal to y |

**Table 1:** Relational Operators

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols. A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a relational operator. There is no such thing as =< or =>.

## Logical operators

There are three logical operators: **and, or**, and **not**. The semantics (meaning) of these operators is similar to their meaning in English. For example, $x > 0$ and $x < 10$ is true only if $x$ is greater than 0 and less than 10.

$n\%2 == 0$ *or* $n\%3 == 0$ is true if either or both of the conditions is true, that is, if the number is divisible by 2 or 3.

Finally, the *not* operator negates a boolean expression, so not $(x > y)$ is true if $x > y$ is $false$, that is, if $x$ is less than or equal to $y$.

Strictly speaking, the operands of the logical operators should be boolean expressions, but Python is not very strict. Any nonzero number is interpreted as $True$:

```
>>> 42 and True
True
```

## Conditional execution

In order to write useful programs, we almost always need the ability to check **conditions** and change the behavior of the program accordingly. **Conditional statements** give us this ability. The simplest form is the *if* statement:

```
if x > 0:
    print('x is positive')
```

The boolean expression after *if* is called the **condition**. If it is true, the indented statement runs. If not, nothing happens.

*if* statements have the same structure as function definitions: a header followed by an indented body. Statements like this are called **compound** statements.

## Conditional execution

There is no limit on the number of statements that can appear in the body, but there has to be at least one. Occasionally, it is useful to have a body with no statements (usually as a place keeper for code you haven't written yet). In that case, you can use the pass statement, which does nothing.

```
if x < 0:
    pass # TODO: need to handle negative values!
```

## Alternative execution

A second form of the *if* statement is **alternative execution**, in which there are two possibilities and the condition determines which one runs. The syntax looks like this:

```
if x % 2 == 0:
    print('x is even')
else:
    print('x is odd')
```

If the remainder when x is divided by 2 is 0, then we know that x is even, and the program displays an appropriate message. If the condition is false, the second set of statements runs. Since the condition must be true or false, exactly one of the alternatives will run. The alternatives are called **branches**, because they are branches in the flow of execution.

## Chained conditionals

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a **chained conditional**:

```python
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

*elif* is an abbreviation of **else if**. Again, exactly one branch will run. There is no limit on the number of *elif* statements.

## Chained conditionals

If there is an *else* clause, it has to be at the end, but there doesn’t have to be one.

```python
if choice == 'a':
    draw_a()
elif choice == 'b':
    draw_b()
elif choice == 'c':
    draw_c()
```

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch runs and the statement ends. Even if more than one condition is true, only the first true branch runs.

## Nested conditionals

One conditional can also be nested within another. We could have written the example in the previous slide like this:

```python
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another if statement, which has two branches of its own. Those two branches are both simple statements, although they could have been conditional statements as well.

## Nested conditionals

Although indentation of statements makes the structure apparent, nested conditionals become difficult to read and is a good idea to avoid them when you can. Logical operators often provide a way to simplify nested conditional statements. For example, we can rewrite the following code using a single conditional:

```python
if 0 < x:
    if x < 10:
        print('x is a positive single-digit number.')
```

The print statement runs only if we make it past both conditionals, so we can get the same effect with the *and* operator:

```python
if 0 < x and x < 10:
    print('x is a positive single-digit number.')
```

For this kind of condition, Python provides a more concise option:

```python
if 0 < x < 10:
    print('x is a positive single-digit number.')
```

## Glossary

**floor division**: An operator, denoted //, that divides two numbers and rounds down (toward negative infinity) to an integer.

**modulus operator**: An operator, denoted with a percent sign (%), that works on integers and returns the remainder when one number is divided by another.

**boolean expression**: An expression whose value is either True or False.

**relational operator**: One of the operators that compares its operands: $==, != , >, <, >=$, and $<=$.

**logical operator**: One of the operators that combines boolean expressions: *and*, *or*, and *not*.

**conditional statement**: A statement that controls the flow of execution depending on some condition.

## Glossary

**condition**: The boolean expression in a conditional statement that determines which branch runs.

**compound statement**: A statement that consists of a header and a body. The header ends with a colon (:). The body is indented relative to the header.

**branch**: One of the alternative sequences of statements in a conditional statement.

**chained conditional**: A conditional statement with a series of alternative branches.

**nested conditional**: A conditional statement that appears in one of the branches of another conditional statement.

## Further Reading

- Chapter 5 of Think Python(2nd Edition) - Allen B. Downey
- Chapter 5 of Python Crash Course - Eric Matthes
- Python Official Website
- Lecture 3 Jupyter Notebook

*Thank you.*
*Any Questions?*