While, While West

Floor division and modulus

The **floor division** operator, `//`, divides two numbers and rounds down to an integer.

```
>>> minutes = 105
>>> minutes / 60
1.75
>>> minutes // 60
1
```

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

```
>>> minutes % 60
45
```

Modulus

The modulus operator is more useful than you might think.

You can check if one number is divisible by another: if `x % y` is 0, then you know that `x` is divisible by `y`.

```
def is_even(number):
  return number % 2 == 0
```

Keyboard input

Python provides a built-in function called `input` that stops the program and waits for the user to type something. When the user presses **Return** or **Enter**, the program resumes and `input` returns what the user typed as a string.

`input` is a fruitful function—it returns what the user typed. You will almost always want to store its return value in a variable.

```
>>> text = input()
What are you waiting for?
>>> text
'What are you waiting for?'
```

You usually want to tell the user what to type. You can pass in a prompt as the argument to `input`.

```
>>> first_number = input('Enter a number: ')
Enter a number: 3
```

Keyboard input

You can use the newline special character, `\n`, if you would like the user to enter input on a new line.

```
>>> first_number = input('Enter a number:\n')
Enter a number:
3
```

Remember that `input` returns the user input as a string. If you want to use the user input for a numerical computation, you need to first convert the return value into the appropriate data type.

```
>>> first_number = input('Enter a number: ')
Enter a number: 3
>>> second_number = input('Enter another number: ')
Enter another number: 2
>>> first_number + second_number
'32'
```

Boolean expressions

A **boolean expression** is an expression that is either true or false.

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

These boolean expressions use the `==` **relational operator**, which produces `True` if the operands are equal and `False` if they are not.

Boolean expressions

There are other relational operators that can be used in boolean expressions.

Expression	Description
`x != y`	x is not equal to y
`x > y`	x is greater than y
`x < y`	x is less than y
`x >= y`	x is greater than or equal to y
`x <= y`	x is less than or equal to y

Relational operators: A couple things to note...

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.

```
>>> x = 2
>>> y = 3
>>> x = y
>>> x
3
```

There is no such thing as `=<` or `=>`.

```
>>> x = 2
>>> y = 3
>>> x =< y
File "<stdin>", line 1
  x =< y
     ^
SyntaxError: invalid syntax
```

Try to remember the phrases "less than or equal to" `<=` and "greater than or equal to" `>=`.

Logical operators

Logical operators are used to *combine* boolean expressions.

In Python, the logical operators are `and`, `or`, and `not`.

The meaning of these logical operators in programming are similar to their meaning in English.

`and`: `x > 0 and x < 10` is true only if `x` is greater than 0 *and* less than 10.

`or`: `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.

`not`: `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`.

Logical operators

`7 > 2 and 7 % 2 == 0` `False`

`8 - 2 < 4 or True` `True`

`not False` `True`

`True or 2 + 3 > x` `True`

`3 % 4 == 0 or (7 % 2 == 1 and 12 % 2 == 1)` `False`

Conditional execution

In almost all useful programs, we need the ability to check conditions and change the behavior of the program accordingly. **Conditional statements** give us this ability.

The simplest conditional statement is the `if` statement:

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

The boolean expression after `if`—`x > 0`—is the **condition.** If it is true, the indented statement runs. If it is false, nothing happens. There is no limit on how many indented statements you can have, but you must have at least one.

Conditional execution

Check if `x` is an even number. If it is, print its value and that it is even.

```
x = 2
```

```
def is_even(number):
  return number % 2 == 0

x = 2
```

Check if `y` is an an odd number. If it is, print its value and that it is odd.

```
def is_even(number):
  return number % 2 == 0

y = 3
```