FLOW CONTROL

CS 3030: Python

Instructor: Damià Fuentes Escoté

University of Colorado
Colorado Springs

Previous lesson – Python basics

- Introduction
- Course description
- Syllabus
- Flow control
  - *Expression >>> 2 + 2*
  - *Precedence >>> (5 - 1) * ((7 + 1) / (3 - 1))*
  - *Common data types >>> string, int and float*
  - *Variables >>> Initialize, overwrite, names,*
  - *First program >>> print(), input(), len(), str(), int()*
  - *Comments >>> One line (#) and multiline (''' ''')*

Flow control

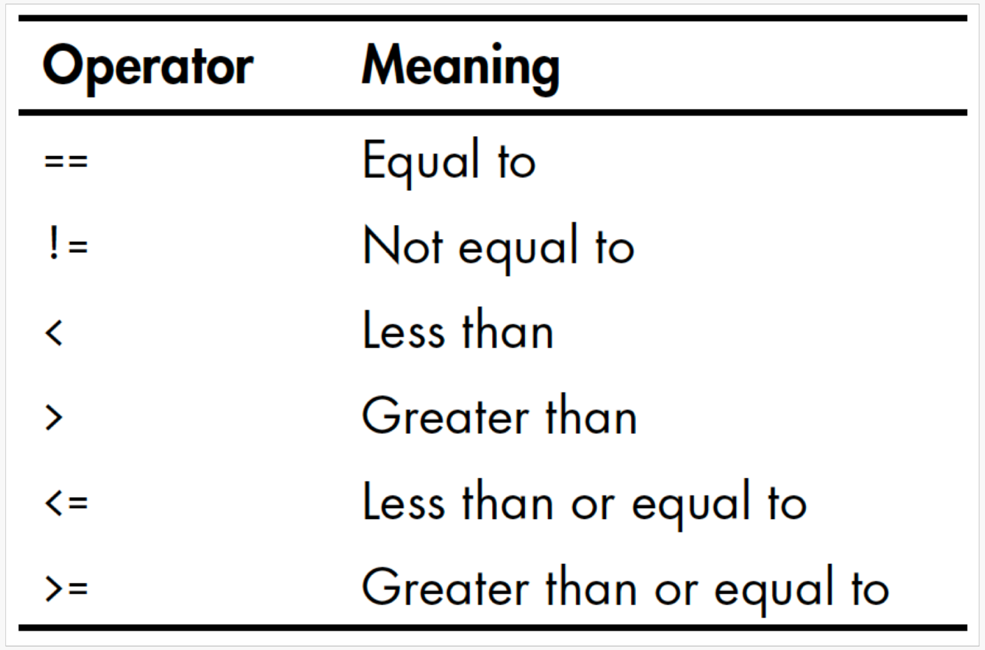
- A program is just a series of instructions.
- **Flow control statements** can decide which Python instructions to execute under which conditions.
- Based on how the expressions evaluate, the program can decide to skip instructions, repeat them, or choose one of several instructions to run.
  - *if, else, elif*
  - *while*
  - *for loops*

Boolean data type

- Only two values:
  - *True*
  - *False*
- spam = True

Comparison Operators

- **Comparison operators** compare two values and evaluate down to a single Boolean value



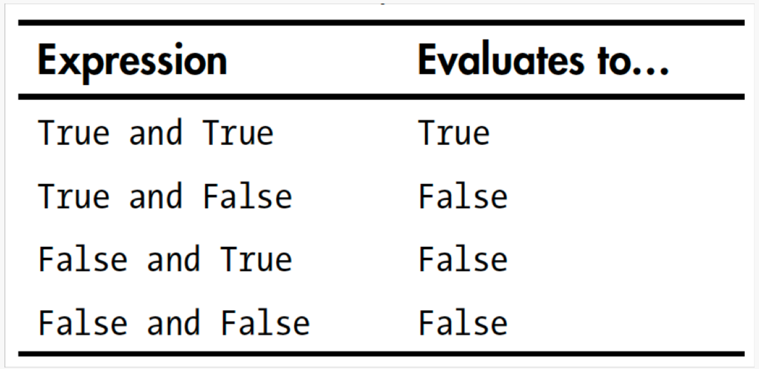
Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

= is different from == :

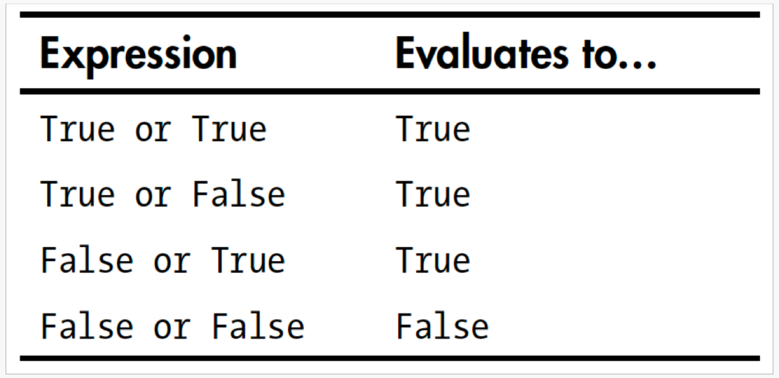
- = is the assignment statement
- == equal to operator

Boolean operators

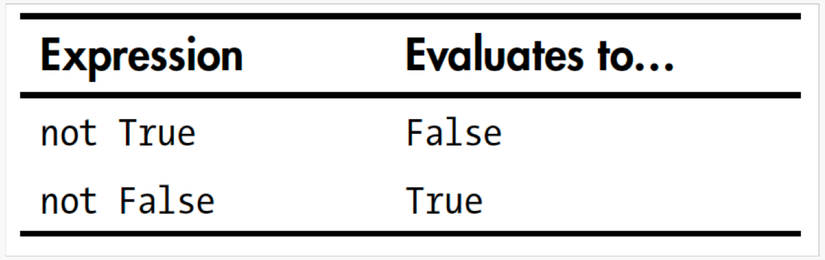
- and, or, and not are used to compare Boolean values.
- Like comparison operators, they evaluate these expressions down to a Boolean value.



Expression	Evaluates to...
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`



Expression	Evaluates to...
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`



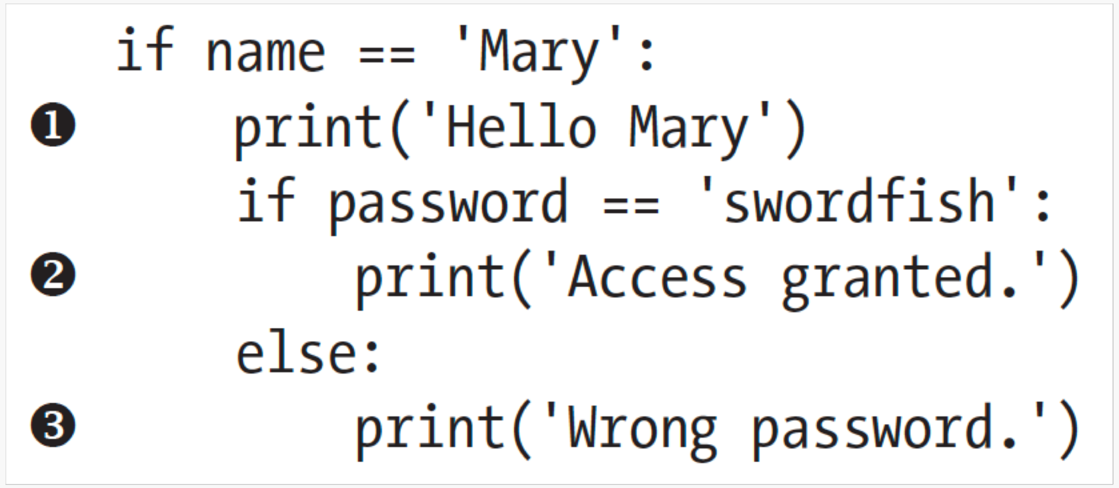
Expression	Evaluates to...
`not True`	`False`
`not False`	`True`

Conditions

- All the expressions with Boolean operators can be considered conditions.
- **Conditions** always evaluate down to a Boolean value, True or False.
  - *(4 < 5) and (5 < 6)   # True*
  - *(1 == 2) or (2 == 2)   # True*

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.



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

if statements

- 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.

```
if name == 'Alice’:
        print('Hi, Alice.’)

print(‘Bye’)
```

else statements

- The **else** clause is executed only when the if statement’s condition is False

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

elif statements

```
if name == 'Alice’:
        print('Hi, Alice.’)
else if age < 12:
        print('You are not Alice, kiddo.’)
```

Same as:

```
if name == 'Alice’:
        print('Hi, Alice.’)
elif age < 12:
        print('You are not Alice, kiddo.’)
```

while loop statements

- The code in a **while** clause will be executed over and over again as long as the while statement’s condition is True .

```
spam = 0
while spam < 5:
        print('Hello, world.’)
        spam = spam + 1
```

break statements

- If the execution in a while reaches a **break** statement, it immediately exits the while loop's clause.

```
while True:
    print('Please type your name.')
    name = input()
    if name == 'your name':
        break
print('Thank you!')
```

continue statements

- When the program execution reaches a **continue** statement, the program execution immediately jumps back to the start of the loop and reevaluates the loop’s condition. (This is also what happens when the execution reaches the end of the loop.)

```
while True:
        print('Who are you?')
        name = input()
        if name != 'Joe':
                continue
        print('Hello, Joe. What is the password? (It is a fish.)')
        password = input()
        if password == 'swordfish':
                break
print('Access granted.')
```

UCCS

“Truthy” and “Falsey” values

- When used in conditions, **0**, **0.0**, and **''** (the empty string) are considered False, while all other values are considered True.

```
numOfGuests = int(input())
if numOfGuests:
        print('Be sure to have enough room for all your guests.’)
else:
        print(‘You don’t have guests.’)
```

for loop statement

- The for **loop** is used to execute a block of code only a certain number of times.
- You can use break and continue statements too.

```
print('My name is’)
for i in range(5):
        print('Jimmy Five Times (' + str(i) + ‘)’)


for i in range(12, 16):

for i in range(0, 10, 2): # Third argument is the step

for i in range(5, -1, -1):
```

Importing modules

- Python comes with a set of modules called the **standard library**. Each module is a Python program that contains a related group of functions that can be embedded in your programs.
- For example, the math module has mathematics related functions, the random module has random number–related functions, and so on.
- Once you import a module, you can use all the cool functions of that module.

```
import random
print(random.randint(1, 10))
```

Importing modules

- Multiple imports

  import *random, sys, os, math*

- from import statements. No need the random. prefix. Two ways:
  - ***from*** *random* ***import*** *randint*

    *print(****randint(1, 10)****)*

  - *from random import* *******

    *print(****randint(1, 10)****)*

Terminate the program early

```
import sys
while True:
        print('Type exit to exit.’)
        response = input()
        if response == 'exit’:
                sys.exit()
        print('You typed ' + response + '.')
```