INTRODUCTION, PYTHON BASICS AND FLOW CONTROL

CS 3080 Python Programming

University of Colorado
Colorado Springs

Credits: Damia Fuentes Escote and Dana Wortman

This lesson

- Syllabus
- Python basics
- Flow control

Intro of Intro

- Yanyan Zhuang
  - *PhD, 2012 yzhuang@uccs.edu*
  - *https://uccs.webex.com/meet/yzhuang*
  - *Office hours*
    - M/W 10:00-10:45am
- TA
  - *Adria Llop Giron´es (allopgir@uccs.edu)*
  - *Office hours*
    - Thursdays, 2:00 – 3:00pm
    - https://uccs.webex.com/meet/allopgir

SYLLABUS

Find it in Canvas

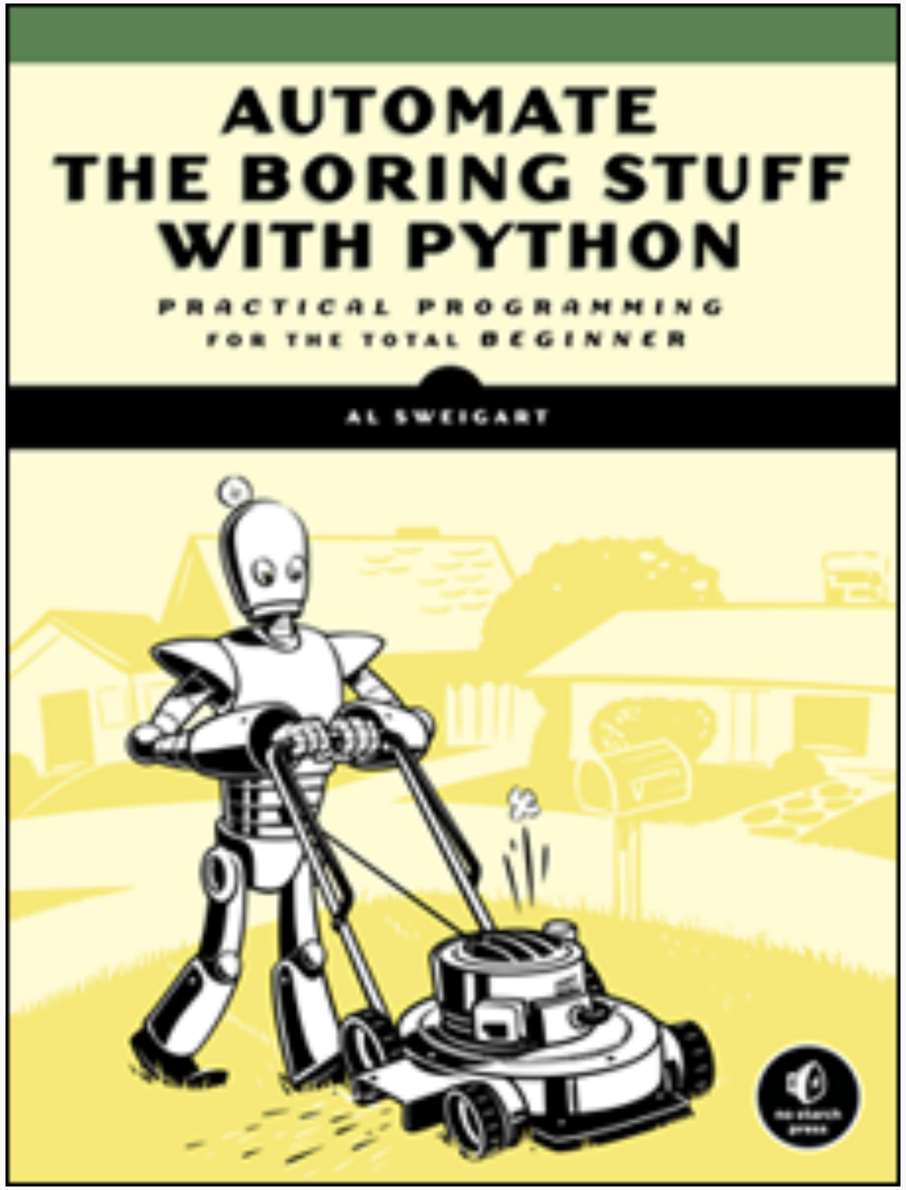
Why this course?

- Python is:
  - *High-level programming language*
  - *Active and supportive community*
  - *Extensive support libraries*
  - *Reliable and efficient*
  - *Accessible (easy to learn and use)*
  - ***High demanded by companies***

Course book

Not all topics that we are going to cover are in the book!

ISBN-10: 1-59327-599-4
ISBN-13: 978-1-59327-599-0


AUTOMATE THE BORING STUFF WITH PYTHON
PRACTICAL PROGRAMMING FOR THE TOTAL BEGINNER
AL SWEIGART

Course content examples – Python basics

```
name = ’’

while name != ‘Yanyan’:
    print('Please type your name.’)
    name = input()

print('Thank you!')
```

What do you notice that’s different from C or Java (at a high level)?

Course content examples – Python basics

```
name = ’’

while name != ‘Yanyan’:
    print('Please type your name.’)
    name = input()

print('Thank you!')
```

What do you notice that’s different from C or Java (at a high level)?

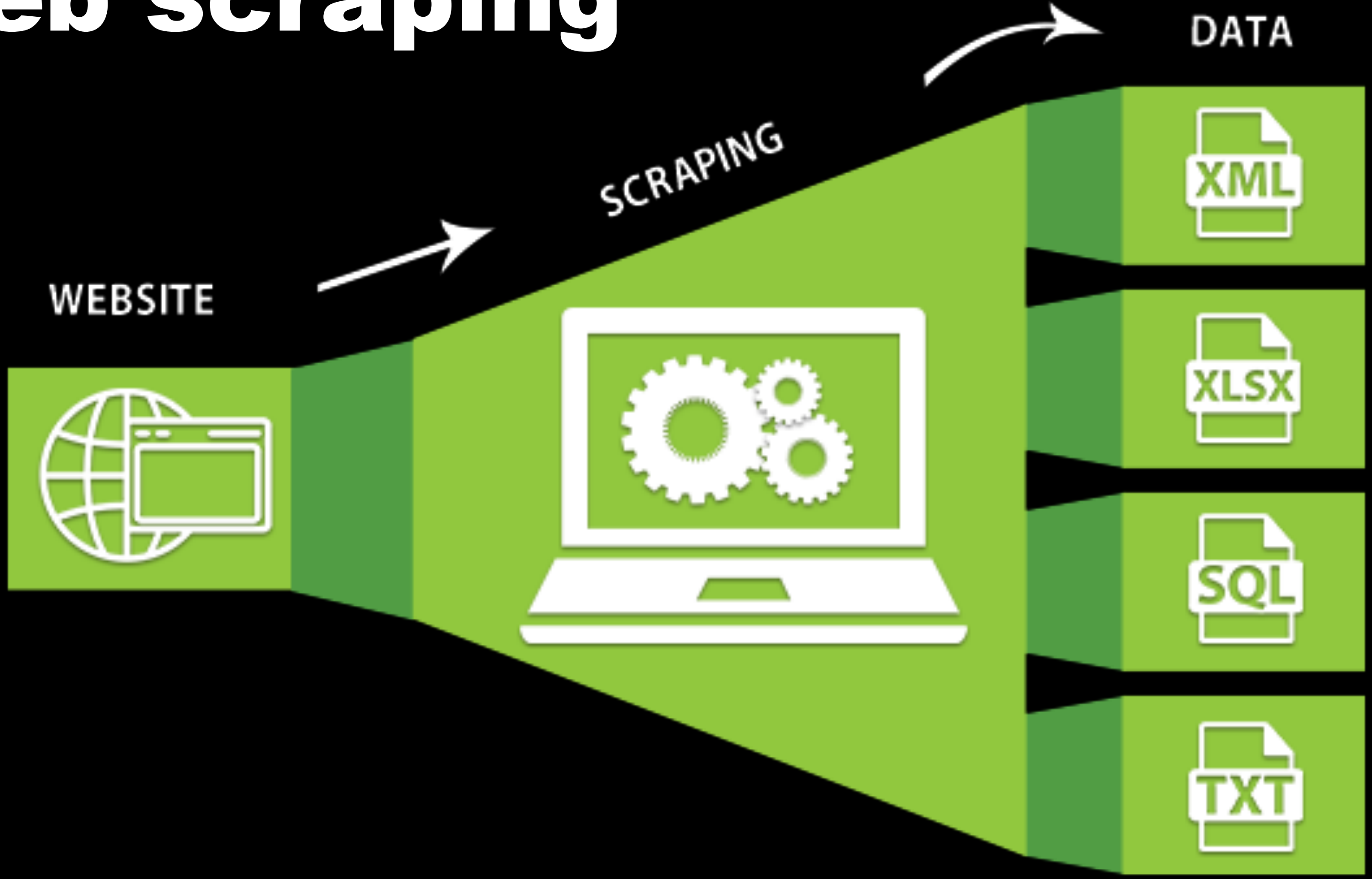
There is no variable declaration in Python! You just assign a value to a variable and it comes into existence. But never use a variable name without having assigned a value

Indentation indicates code blocks

Course content examples – Advanced python

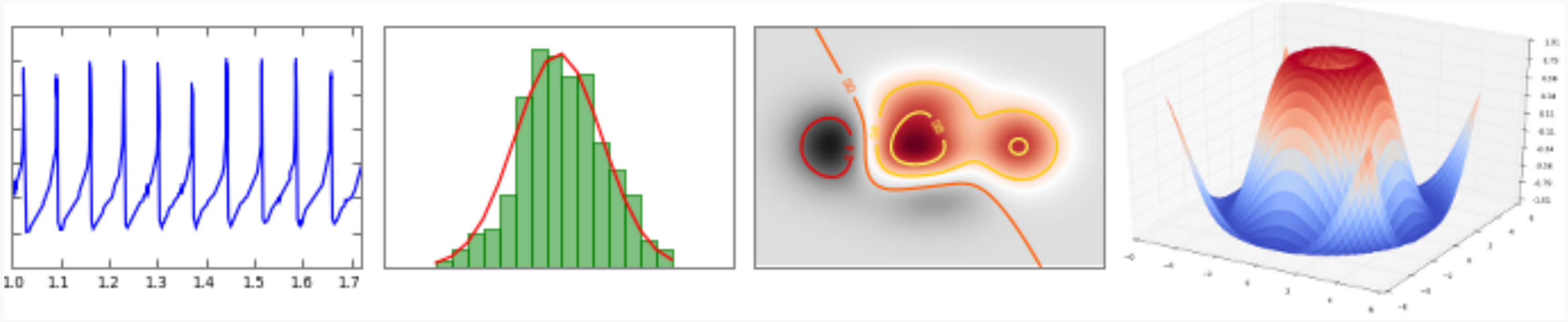
```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_whee():
    print("Whee!")
```

Course content examples – Web scraping



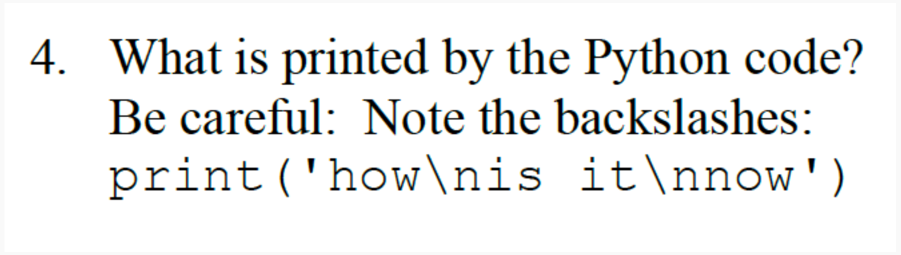
Course content examples – Send email and text messages

Course content examples – Data visualization with matplotlib



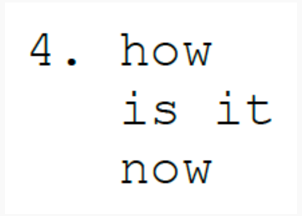
Quizzes

- 5-10 minutes quiz of the past topics
- 8% of the total, but not graded
- Individual

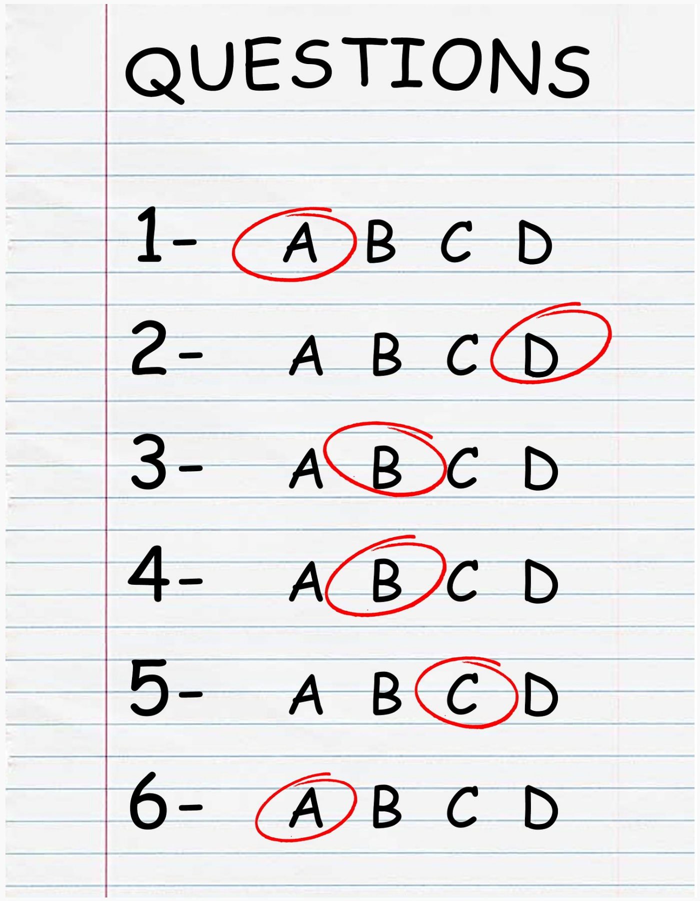


4. What is printed by the Python code? Be careful: Note the backslashes:

```
print('how\nis it\nnow')
```



```
4. how
   is it
   now
```



Midterm

- Midterm, 10% of the total (mid to late October)
- Individual
- In class
- **Laptop required (online in Canvas, open book)**
- Similar to quizzes and homework

Final exam

- Topics about lectures, quizzes and homework.
- **Monday December 14th, 10:20am to 12:20pm** –17% of the total
- Individual.
- In-class.
- **Laptop required (online in Canvas, open book)**

Final project

- 15% of the total
- Develop whatever you want! But in Python.
- Groups of 2 or 3.
- One-page **project proposal** by **9/23** (about a month from now)
- **Report** of 3 – 5 pages double space 12 points by **12/14**
- All team members must collaborate in the **GitHub** project. **A member that has not collaborated in the project or that has written just a few lines will receive a 0**
- If you need a partner, start a discussion in Canvas https://community.canvaslms.com/t5/Student-Guide/How-do-I-create-a-course-discussion-as-a-student/ta-p/300

Final project

Project can be based on a topic not covered in class

- We recommend you think in developing something that can be used by the community later on
- Do not develop a UI based project. You can have UI in your project but that should not be the whole point of the project
- Developing a game, something with hardware or a full project using Selenium may be accepted but not recommended

Grading

Category	Percentage
Homework	50%
Quizzes	8%
Final project	15%
Midterm	10%
Final exam	17%

Cheating

- Plagiarism is a serious offense and all involved parties will be penalized according to the Academic Honesty Policy.
- I will report to the Dean's Office and they will decide.
  - *I will not be able to control the consequences.*
- **If you copy something like a method, a class, etc. from the internet, reference it!**
  - *How many times have copied a piece of code from the internet and pasted it in your code?*
  - *Make sure it does what you expect it to do*

My last advice…

- If you don’t know how to do something in a homework/project, Google it before you ask the question
- http://letmegooglethat.com/

PYTHON BASICS

Interactive shell

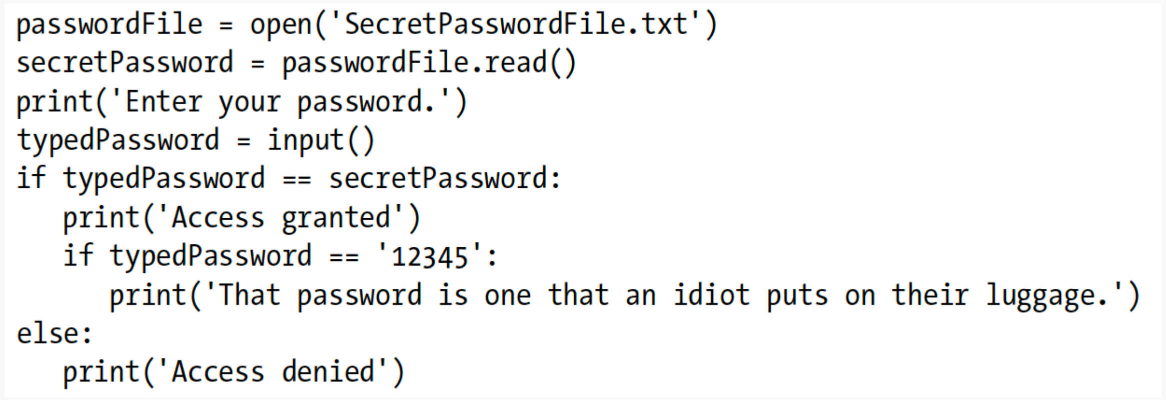
- A shell is a program that lets you type instructions into the computer.
- Python’s interactive shell lets you enter statements for the Python interpreter software to run. The computer reads the statements you enter and runs them immediately.
- From the Command Line (Windows) or Terminal (Mac) write (or from PyCharm):

```
$ python


>>> print('Hello world!’)
Hello world!
```

Python interpreter

- An interpreter is a program that reads and executes code.
- Python interpreter software reads source code (written in the Python language) and performs its statements.

Source code



```
passwordFile = open('SecretPasswordFile.txt')
secretPassword = passwordFile.read()
print('Enter your password.')
typedPassword = input()
if typedPassword == secretPassword:
   print('Access granted')
   if typedPassword == '12345':
      print('That password is one that an idiot puts on their luggage.')
else:
   print('Access denied')
```

Virtual Environments

- What is a Virtual Environment?
  - *Create an isolated environment for Python projects*

- Why?
  - *Each project have its own dependencies, so don’t install all those packages (or modules) to your system*

- How?

  $ pip install venv (or virtualenv)

  $ python3 -m venv env

  $ source env/bin/activate

  (env) $ deactivate

Try
$ which python
(env) $ which python

Python Software

An integrated development environment (IDE) is a software suite that consolidates basic tools required to write and test software.

They use the python interpreter!

Recommended:

- **PyCharm Community Edition IDE**
  - *https://www.jetbrains.com/pycharm/*

Others:

- Visual Studio IDE
- Sublime text + Command line
- IDLE (used in the book)
- Online Python Editor and IDE (https://repl.it/languages/python3)
- etc

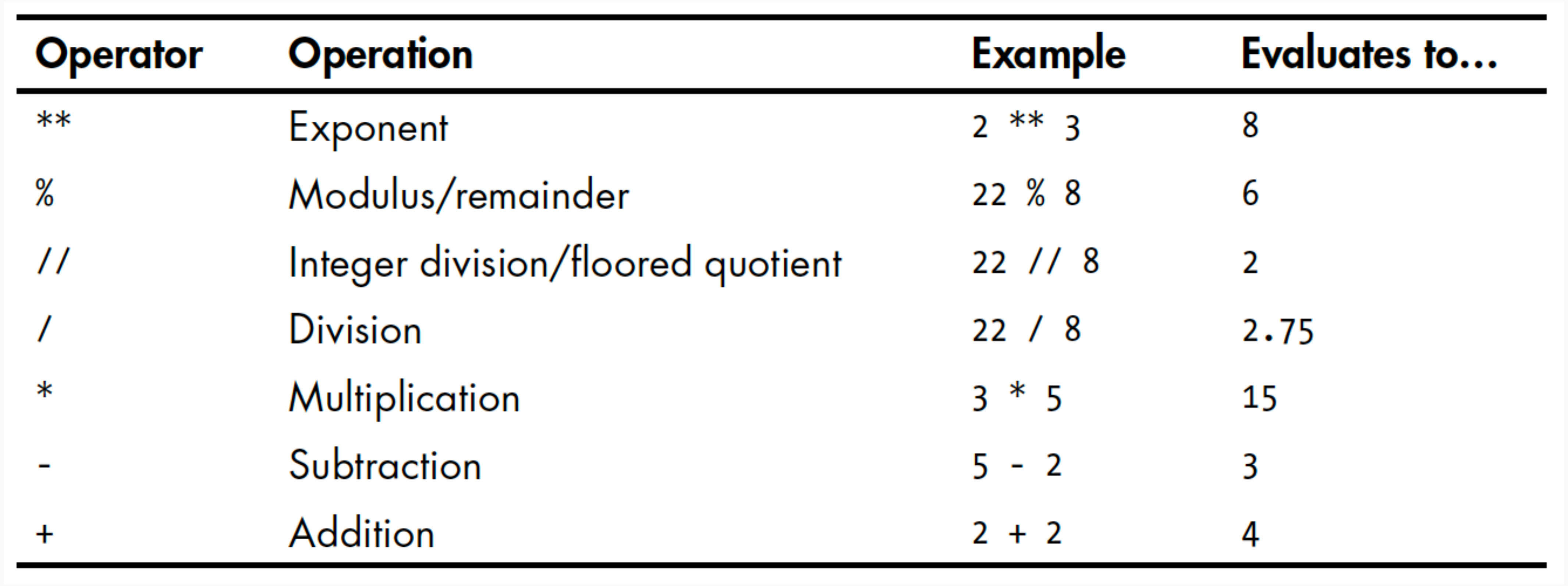
Hardware
Any computer!

Python basics

- **Expression**
  - *Combinations of* ***values*** *and* ***operators*** *that can* ***evaluate*** *down to a single value*
  - *2 + 2*
  - *4 * 3*
  - *5 – 4 * (3 + 1)*
  - *2*

- 2 + 2 is evaluated down to a single value, 4.
- A single value with no operators is also considered an expression, though it evaluates only to itself.

Python basics



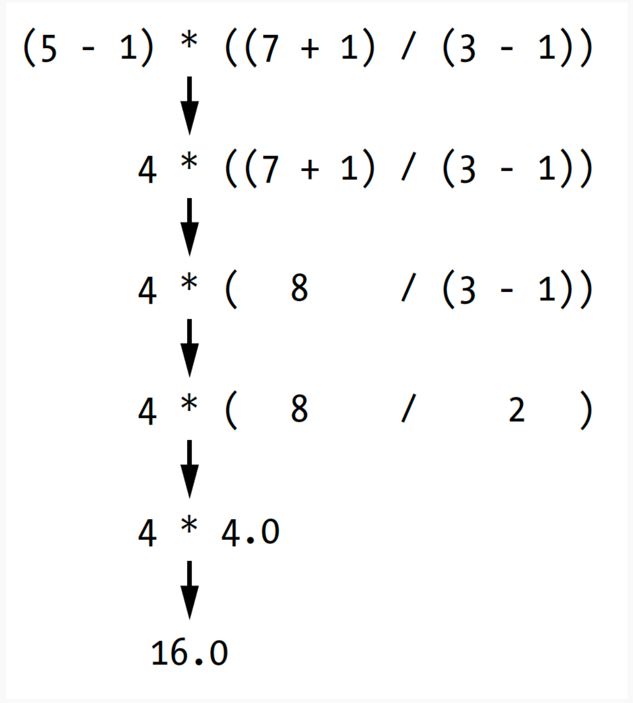
Operator	Operation	Example	Evaluates to...
**	Exponent	`2 ** 3`	8
%	Modulus/remainder	`22 % 8`	6
//	Integer division/floored quotient	`22 // 8`	2
/	Division	`22 / 8`	2.75
*	Multiplication	`3 * 5`	15
-	Subtraction	`5 - 2`	3
+	Addition	`2 + 2`	4

Python basics

- **Precedence**
  - *The ** operator is evaluated first;*
  - *the * , / , // , and % operators are evaluated next, from left to right;*
  - *the + and - operators are evaluated last (also from left to right).*
  - *You can use parentheses ( ) to override the usual precedence if you need to.*

Python basics

- **Precedence**
  - *Python will keep evaluating parts of the expression until it becomes a single value*



```
(5 - 1) * ((7 + 1) / (3 - 1))
        ↓
      4 * ((7 + 1) / (3 - 1))
        ↓
      4 * (   8    / (3 - 1))
        ↓
      4 * (   8    /    2   )
        ↓
      4 * 4.0
        ↓
       16.0
```

Python basics

- **Common data types**
  - *A data type is a category for values, and every value belongs to exactly one data type*
  - *The meaning of an operator may change based on the data types of the values next to it.*
    - String replication operator
      - *'Alice' * 5*

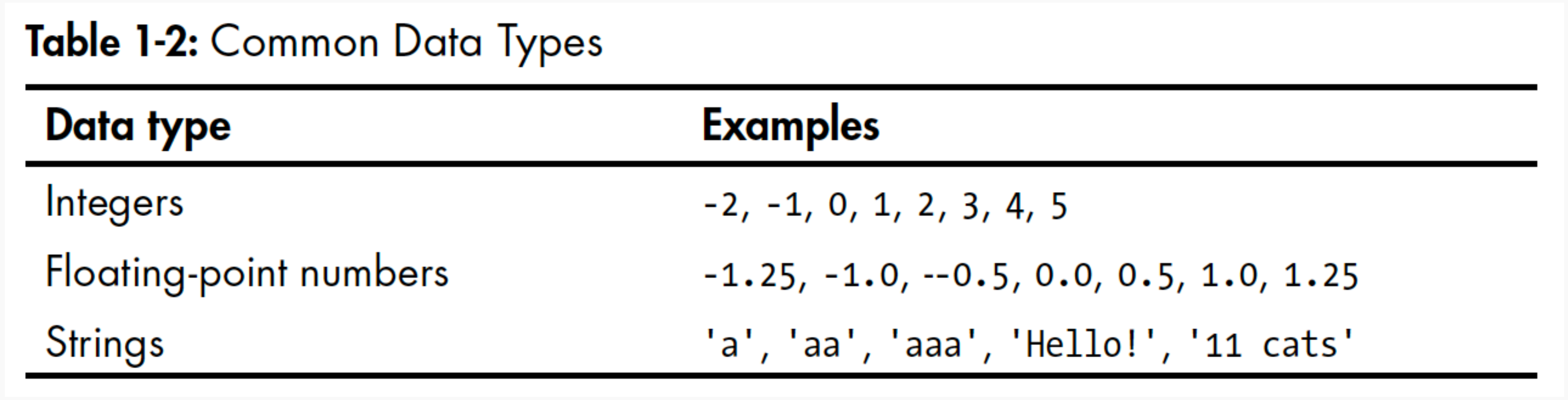


Table 1-2: Common Data Types

Data type	Examples
Integers	`-2, -1, 0, 1, 2, 3, 4, 5`
Floating-point numbers	`-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25`
Strings	`'a', 'aa', 'aaa', 'Hello!', '11 cats'`

Storing values in variables

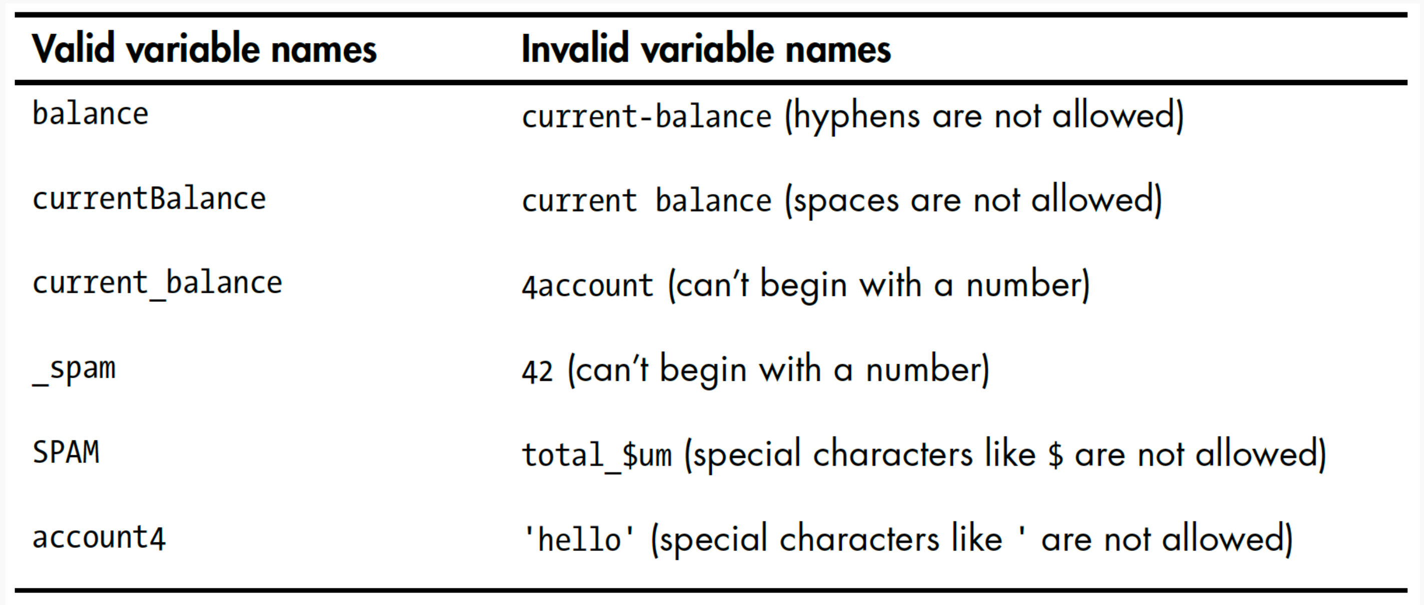
- You’ll store values in variables with an **assignment statement**. An assignment statement consists of a variable name, an equal sign (called the **assignment operator**), and the value to be stored.
- A variable is **initialized** (or created) the first time a value is stored in it
- **Overwriting** the variable -> When a variable is assigned a new value, the old value is forgotten. It doesn’t matter the data type.
- **Multiple assignment**
  - *a = b = c = 1*
  - *a,b,c = 1,2,"john"*

I do NOT recommend using multiple assignment! It can be confusing and lead to mistakes!

Variable names

Underscores are a special-use character in classes, so don't use them to start variable names!

- 3 rules:
  - *It can be only one word, i.e., no spaces.*
  - *It can use only letters, numbers, and the underscore (_) character.*
  - *It can't begin with a number.*



Valid variable names	Invalid variable names
`balance`	`current-balance` (hyphens are not allowed)
`currentBalance`	`current balance` (spaces are not allowed)
`current_balance`	`4account` (can't begin with a number)
`_spam`	`42` (can't begin with a number)
`SPAM`	`total_$um` (special characters like $ are not allowed)
`account4`	`'hello'` (special characters like ' are not allowed)

Variable names

- Variable names are **case-sensitive**
  - *spam , SPAM , Spam , and sPaM are four different variables.*
- It is a Python convention to start your variables with a lowercase letter.
- Descriptive names makes your code more readable.
- **Camelcase**
  - *lookLikeThis*
- **Underscores** (official Python code style)
  - *looking_like_this*
- It doesn’t matter which way, but be consistent in your code!

Comments

- Python ignores comments, and you can use them to write notes or remind yourself what the code is trying to do.
- Any text for the rest of the line following a hash mark (#) is part of a comment.
  - *# This is a one line comment*
- Any text in between three delimiters (''') is part of a comment. Used to explain things in more detail.
  - *'''*
    This is a multiline comment.
    Second line.
    '''

Basic functions

- The **print()** function displays the string value inside the parentheses on the screen.
- The **input()** function waits for the user to type some text on the keyboard and press enter. Always returns a string.
- The **len()** function evaluates to the integer value of the number of characters in the string argument (or a variable containing a string).
  - *len('hello world') evaluates to 11*
- The **str()** function can be passed an integer value and will evaluate to a string value version of it.
  - *str(29) evaluates to '29'*
- The **int()** function can be passed an string value and will evaluate to a int value version of it.
  - *int('29') evaluates to 29*
  - *The int() function is also useful if you need to round a floating-point number down*

First program

- Ask the user for his name and age.
- Then, tell him:
  - *The length of his name*
  - *How old will be in a year (= age + 1)*

- https://repl.it/languages/python3

```
print('Hello world!')

# ask for their name
print('What is your name?')
myName = input()
print('It is good to meet you, ' + myName)

# print the length of their name
print('The length of your name is:')
print(len(myName))

# ask for their age
print('What is your age?')
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.'
```

Why do we need to convert an int to a string?

FLOW CONTROL

Flow control

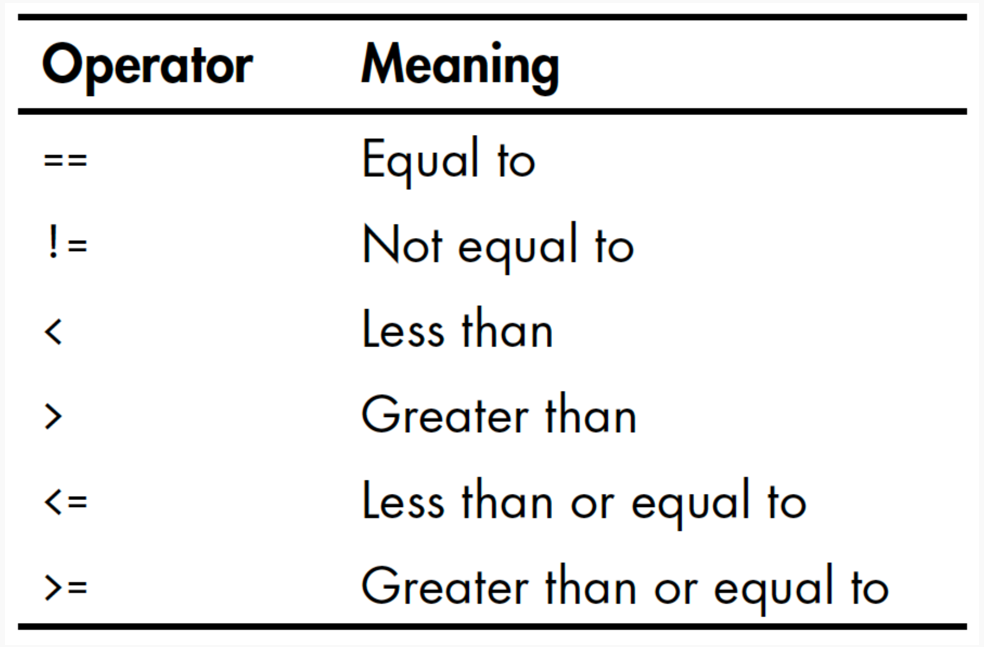
- A program is just a series of statements.
- **Flow control statements** can decide which Python instructions to execute under which conditions.
- Based on how the expressions evaluate, the program can decide to skip instructions, repeat them, or choose one of several instructions to run.
  - *if, else,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

Boolean data type

- Only two values:
  - *True*
  - *False*
- spam = True
- is_over_21 = False

Comparison Operators

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

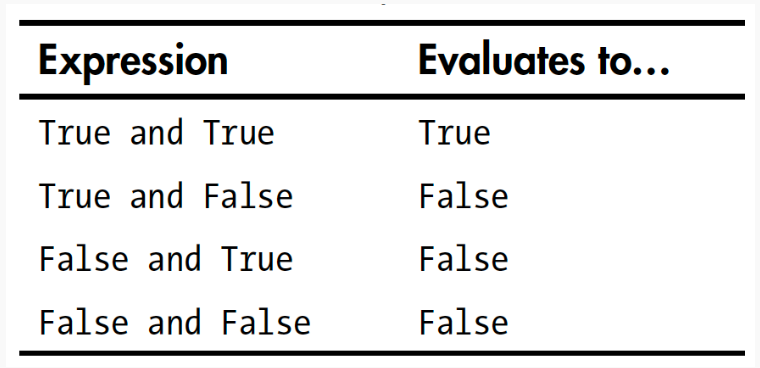


Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

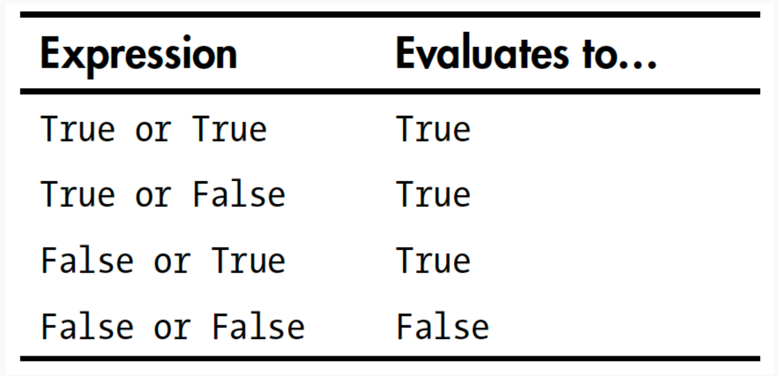
= is different from == :
- = is the assignment statement
- == equal to operator

Boolean operators

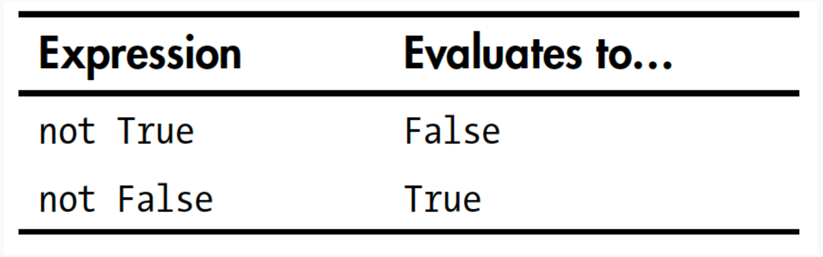
- and, or, and not are used to compare Boolean values.
- Like comparison operators, they evaluate these expressions down to a Boolean value.



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.')
```

In most editors…

Highlight code blocks and press TAB to indent

Press SHIFT-TAB to un-indent

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.')
```

“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 do not 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):

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 to add 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 + '.')
```

Lecture recordings

- 8/24 Lecture
  https://uccs.webex.com/recordingservice/sites/uccs/recording/playback/7eb3b13b26b64f7cb12d1562bfc77749