# Introduction to CM
# Introduction to Pyt

## Week 1

Sanja Damjanovic

# Week 1 Outline

- Introduction to CMPE180A
- Getting to know everyone
- Python overview
- Tutorial

# Sanja Damjanović, PhD

## CMPE 180A Instructor

- Dipl.-ing EE, MSc EE @ University of Belgrade, Serbia
- PhD EE @ University of Twente, The Netherlands
- Expertise:
  - Deep learning, Computer vision, Digital signal processing, Image proce
- Work Experience:
  - Ruhr University Bochum (DE), Wageningen University and Research (NL
  - applied researcher: telecommunications, agro-technology, consumer re
  - university lecturer: deep learning, computer vision, digital signal proces
- Countries:
  - USA, The Netherlands,  Germany, Serbia
- Interests:
  - generative AI, multimodal LLMs,  computer vision, teaching, public spe

# CMPE 180A Calendar

## Off-campus, meetings via Zoom

- Lectures:
  - 5.30 pm - 9.30 pm PT on Tuesdays
  - 07/15/2025 through 09/23/2025
- Office hours: 7pm-8pm PT on Mondays
  - Please try to join on time. If you plan to c
  - I strongly encourage attending to get valu assignments

# CMPE 180A Course Sche

| Week # | Date | Topics |
|---|---|---|
| 1 | 7/15/2025 | Introduction to CMPE180A<br>Introduction to Python, built-in datatypes, conditional expressions, comprehension syntax |
| 2 | 7/22/2025 | Control Flow, Functions, Recursion, Algorithm Analysis, Object-Oriented Programming |
| 3 | 7/29/2025 | Array-based sequences, Stacks, Queues, Deques, Linked List |
| 4 | 8/5/2025 | Trees and Tree Algorithms |
| 5 | 8/12/2025 | Midterm, Project Introduction |

| Week # | D |
|---|---|
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |

# Grading

- Credit (CR) or No Credit (NC)

| Grade | Grade |
| --- | --- |
| Credit (CR) | 84-100 |
| No credit (NC) | 0-83 |

# Assignments and Quizzes

- 4 homework assignments (4x 10 points)
- Midterm (15 points)
- The comprehensive final (30 points)
- Team term project (15 points)

# Assignment Deadlines

- Homework assignment submission deadline
  - ~2 weeks
- Deadline extension
  - Possible due to individual circumstances
  - Request individual deadline extension via sanja.damjanovic@sjsu.edu
  - Max 3 extension tokens

# Resources and Reading M


[1] **Problem Solving with Algorithms and Data Structures using Pytho**
Luther College, freely available online via
https://runestone.academy/runestone/books/published/pythonds3/index.ht

[2] **Data Structures and Algorithms in Python**,
by Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, availa
https://www.proquest.com/legacydocview/EBC/4946360?accountid=10361

[3] **Python Cookbook**, 3rd Edition, David Beazley, Brian K. Jones (Free a
https://www.oreilly.com/)

[4] **Think Python**, 3rd edition, Allen B. Downey, freely available online via
https://allendowney.github.io/ThinkPython/

# Lectures

- Recap+
- New topic introduction
- In-class quiz questions for understanding check
- Tutorial
- In-class assignments
- 2 breaks:  15min @~6:30pm, 10min @~8.00pm
- Real-time online attendance recommended
- Lecture recordings available in Canvas, see PANOPTO
- Slides available before the beginning of the lecture in Canvas
- Check announcements regularly

# Introduce yourself

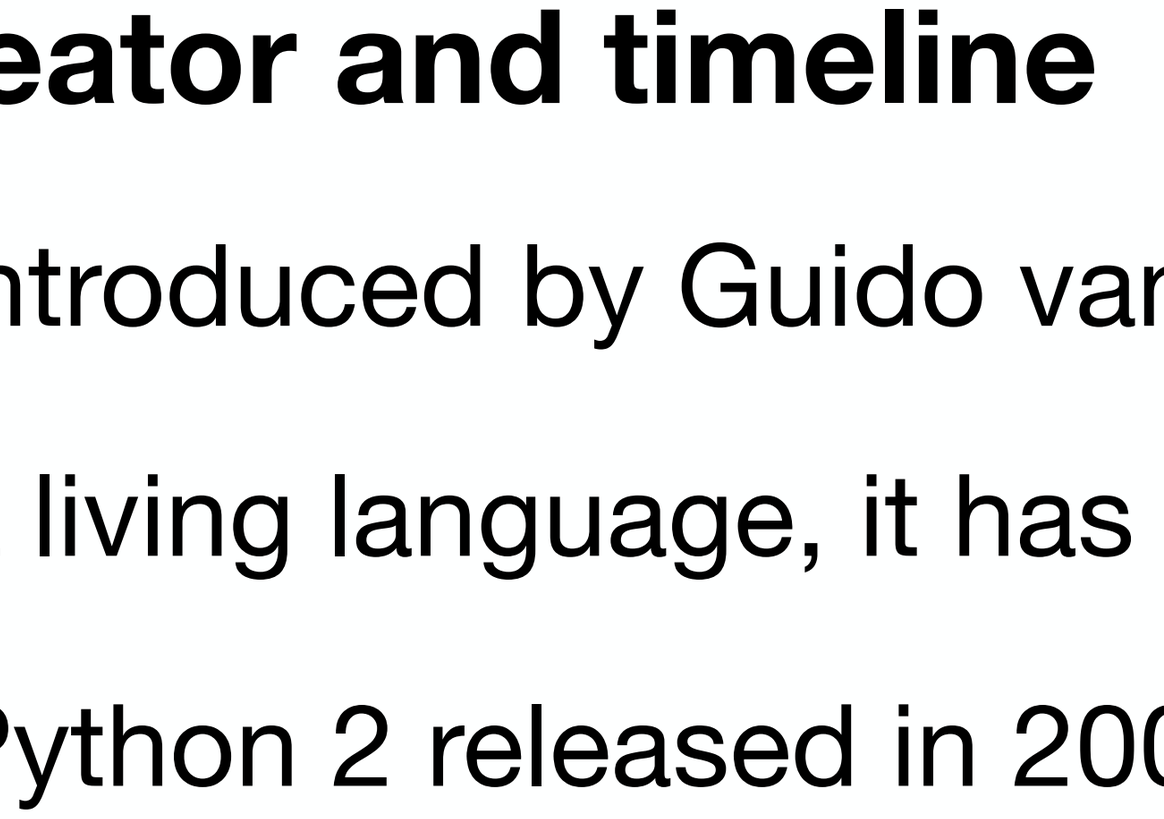
- Where do you live
- Your educational and work background
- Share your hobby or  something interesting
- Your experience with programming and you

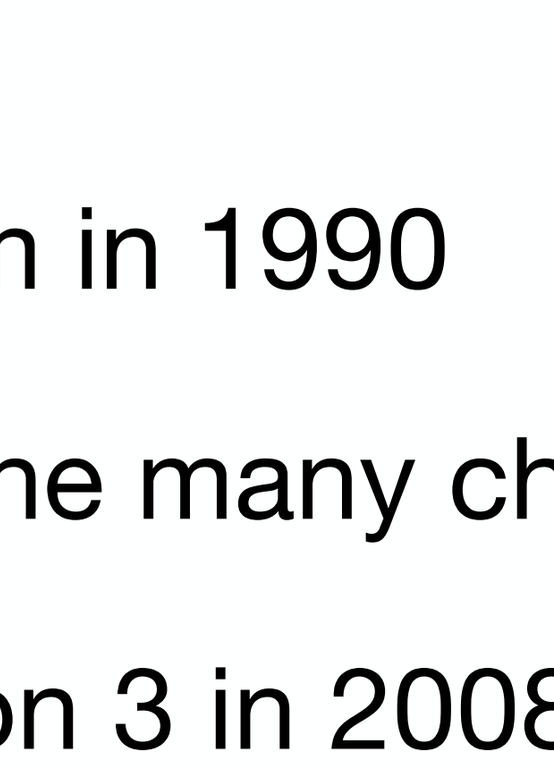
# General Introducti

# Python

## Creator and timeline

- introduced by Guido van Rossum in 1990
- a living language, it has undergone many ch
- Python 2 released in 2000, Python 3 in 2008




02/25/2019

# Python

- Python:
  - https://www.python.org/
- Python Documentation
  - https://www.python.org/doc/
- Python Enhancement Proposals (PEPs)
  - https://peps.python.org/

# Python Enhancement Pro

## the style guide for how to format Pyth

peps.python.org/pep-0008/



### Contents



# PEP 8 – Style Guide

**Author:** Guido van Rossum <gui
Coghlan <ncoghlan at g

**Status:** Active
**Type:** Process
**Created:** 05-Jul-2001
**Post-History:** 05-Jul-2001, 01-Aug-20

▶ **Table of Contents**

## Introduction

This document gives coding conventions
Python distribution. Please see the comp
the C implementation of Python.

This document and PEP 257 (Docstring
essay, with some additions from Barry's

This style guide evolves over time as ad
obsolete by changes in the language itse

Many projects have their own coding sty
guides take precedence for that project.

# Programming languages

## In general

- Low level vs high level
  - program instructions on the machine level or
- General vs targeted to application domain
  - widely applicable or are fine-tuned to a dom
- Interpreted versus compiled
  - refers to whether the sequence of instruction
    source code, is executed directly (by an inte
    converted (by a compiler) into a sequence of

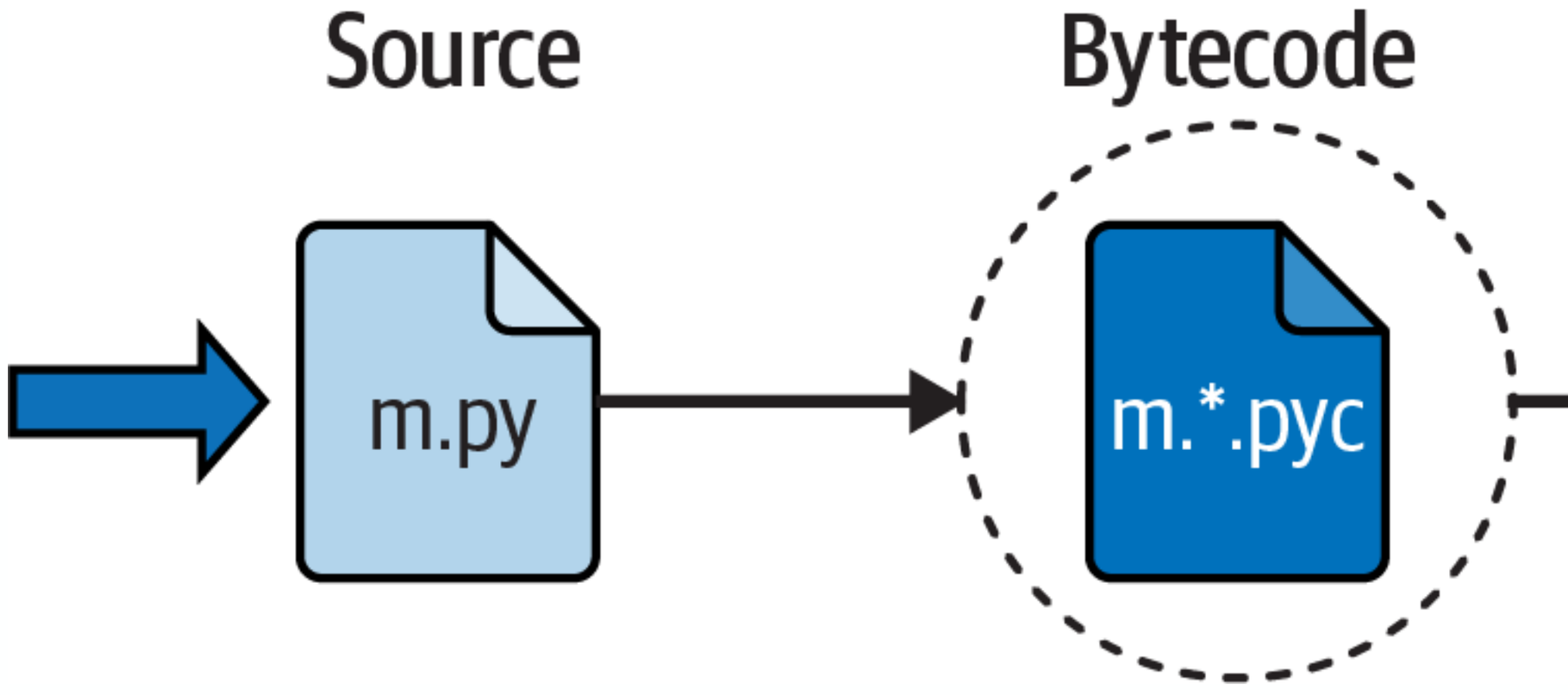
# Python

- Low level vs **high level**
- **General** vs targeted to application domain
- **Interpreted** vs compiled
- Python is a general-purpose programming l
  effectively to build almost any kind of progra
  access to the computer’s hardware.

# Python

- Easy to get started with
- Readable and concise syntax
- Rich standard library and built-in data struct
- Interactive, interpreted, cross-disciplinary
- Rapid prototyping
- Application areas:  Data science, machine le
  Cloud/Admin, web development,…

# Python’s traditional execu




- Bytecode: a lower-level, platform-independe code, ‘compiled’ to or loaded from *.pyc* files
- PVM: Python Virtual Machine executes the c

# Python

- Rapid prototyping
- Numeric and scientific programming
- Object oriented language
- Free & open
- Portable

# Python Disadvantage

- Python’s *execution speed* may not always b
  and lower-level languages such as fully com

# Python Installation

- Anaconda (anaconda.org ):
  - Pre-compiled meta-distribution
  - Includes many scientific libraries
  - Can create environments
- python.org
  - Install libraries

# Python 3

## https://www.python.org/

- Install and run on your computer Python (+IDE):
  - Command line https://www.python.org/downloads/ (p
  - Anaconda https://www.anaconda.com/
  - JupyterLab (notebook) https://jupyter.org/install
  - VS Code (https://code.visualstudio.com/)
  - Google Colab https://colab.google/
- File extensions: *.py, *.ipynb
- A module is a .py file containing Python definitions and
- Modules are imported using ‘import’ statement

# Python program

## *.py

- A Python program (script, source code):
  - a sequence of definitions and commands (s
  - a plain text file with suffix .py
- executed by the Python interpreter in somethi
- Objects are of certain type e.g. int/float
- Objects and operators can be combined to fo
  - Expression evaluates to an object of some

# Python libraries/packages

- Import Python libraries/packages/modules using:
- Module:
  - *.py file
- Packages:
  - numPy: Scientific computing
  - pandas: open source data analysis and manipu
  - scikit-learn: machine learning
  - SciPy, matplotlib, …

Pytho

# Python

- Syntax relies on the use of white spaces
  - An individual statement ends with a newli
- Comments are ignored by the Python interp

# Name (identifier)

## Identifiers, Objects, and the Assignm

- Assignment statement:
  - temperature = 98.6
  - bindings of names/identifiers to objects


- Names/identifiers: case-sensitive, cannot begin with
- The identifier **temperature** is associated with an ins

# Identifiers in Pytho

- 33 specially reserved words tha identifiers:

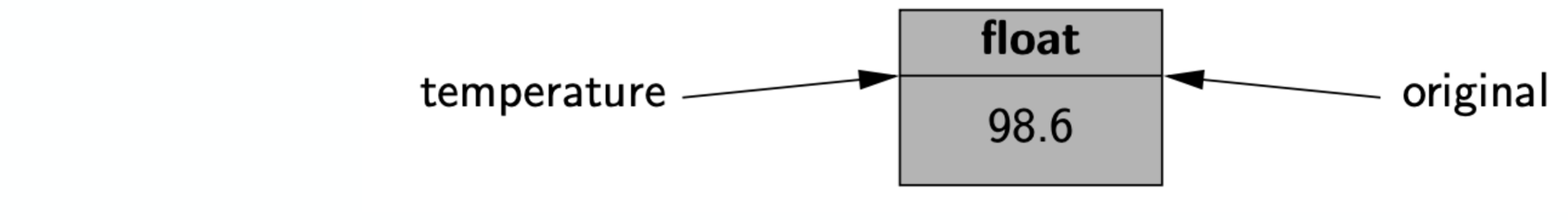
| Reserved Words | | | | | |
|---|---|---|---|---|---|
| False | as | continue | else | from | i |
| None | assert | def | except | global | i |
| True | break | del | finally | if | l |
| and | class | elif | for | import | r |

# Aliases and Reassignmen

- Identifiers *temperature* and original are alias

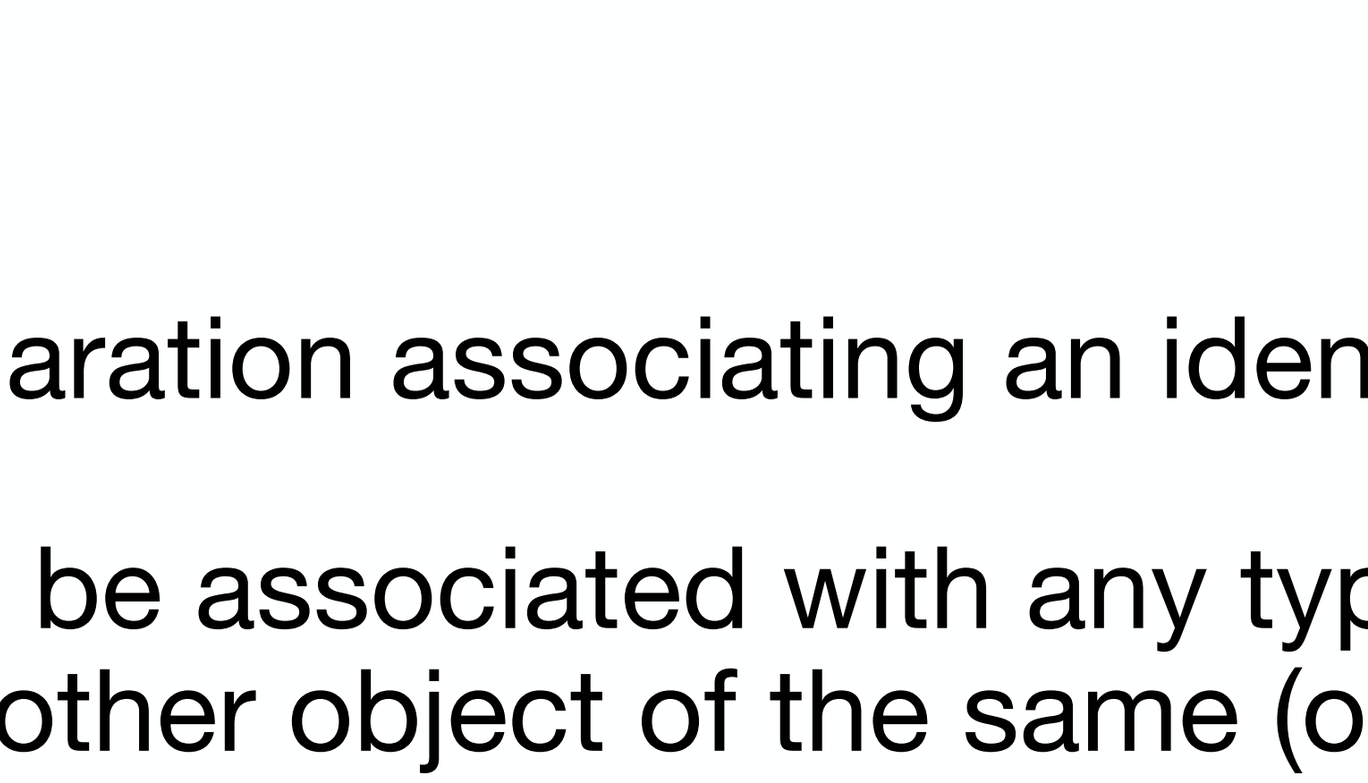

- *temperature = temperature + 5.0*

- The temperature identifier has been assigne
continues to refer to the previously existing



# Dynamically typed langua

- No advance declaration associating an iden
- An identifier can be associated with any typ reassigned to another object of the same (o



# Python's built-in cla

- the **int** class for intege
- the **float** class for float
- the **str** class for charac

—> Tutorial

# An Example Progra

```
print('Welcome to the GPA calculator.')
print('Please enter all your letter grades,
print('Enter a blank line to designate the e
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33,
          'C+':2.33, 'C':2.0, 'C':1.67, 'D+':1.33,
num_courses = 0
total_points = 0
done = False
while not done:
  grade = input( )                                  # re
  if grade == '':                                   # en
    done = True
  elif grade not in points:                         # u
    print("Unknown grade '{0}' being ignored"
  else:
    num_courses += 1
    total_points += points[grade]
if num_courses > 0:                                 # av
  print('Your GPA is {0:.3}'.format(total_points
```

# Objects

- The process of creating a new instanc
  **instantiation**.
- To instantiate an object we usually inv
  class:

```
w = Widget()
```

  - This is assuming that the constructor do
- If the constructor does require param
  such as

```
w = Widget(a, b, c)
```

- Many of Python's built-in classes a **lit**
  new instances. For example, the com

```
temperature = 98.6
```

  results in the creation of a new instan

# Calling Methods

- Python supports functions a synta
  **sorted(data)**, in which case dat
  the function.
- Python's classes may also define
  (also known as member function
  a specific instance of a class usin
- For example, Python's list class h
  that can be invoked with a syntax
  - This particular method rearranges the
    they are sorted.

# Built-In Classes

| Class | Description |
|---|---|
| **bool** | Boolean value |
| **int** | integer (arbitrary magnitud |
| **float** | floating-point number |
| **list** | mutable sequence of objec |
| **tuple** | immutable sequence of ob |
| **str** | character string |
| **set** | unordered set of distinct ob |
| **frozenset** | immutable form of set clas |
| **dict** | associative mapping (aka d |

- A class is immutable if each has a fixed value upon insta subsequently be changed. F **float** class is immutable.

# The bool Class

- The **bool** class is used for logica
  the only two instances of that cl
  literals:

  True and False

- The default constructor, bool( ),
- Python allows the creation of a B
  nonboolean type using the synta
  The interpretation depends upon
  parameter.
  - Numbers evaluate to False if zero, an
  - Sequences and other container types
    evaluate to False if empty and True if

# The int Class

- The **int** class is designed to repr
  arbitrary magnitude.
  - Python automatically chooses the inte
    integer based upon the magnitude of
- The integer constructor, int( ), re
- This constructor can also constru
  upon an existing value of anothe
  - For example, if f represents a floating
    produces the truncated value of f. For
    the value 3, while int(−3.9) produces
  - The constructor can also be used to p
    an integer. For example, the expressio
    integer value 137.

# The float Class

- The **float** class is the floating-
  - The floating-point equivalent of an expressed directly as 2.0.
  - One other form of literal for floating notation. For example, the literal 6. mathematical value $6.022 \times 10^{23}$.
- The constructor float( ) returns
- When given a parameter, the c the equivalent floating-point va
  - float(2) returns the floating-point v
  - float('3.14') returns 3.14

# The list Class

- A **list** instance stores a sequence of references (or pointers) to objects
- Elements of a list may be arbitrary ob object).
- Lists are array-based sequences and elements indexed from 0 to n−1 inclu
- Lists have the ability to dynamically e capacities as needed.
- Python uses the characters [ ] as del
  - [ ] is an empty list.
  - [‘red’, ‘green’, ‘blue’] is a list containing three
- The list( ) constructor produces an e
- The list constructor will accept any it
  - list(‘hello’) produces a list of individual chara

# The tuple Class

- The **tuple** class provides an (unchangeable) version of a allows instances to have an representation that may be than that of a list. Parenthes
  - The empty tuple is ()
- To express a tuple of length comma must be placed afte within the parentheses.
  - For example, (17,) is a one-elem

# The str Class

- String literals can be en
  quotes, as in ‘hello’, or
  in "hello".
- A string can also begin
  three single or double q
  contains newlines in it.

```
print("""Welcome to the GPA
Please enter all your letter gr
Enter a blank line to designat
```

# The set Class

- Python's **set** class represents a s
  elements, without duplicates, an
  order to those elements.
- Only instances of immutable typ
  Python set. Therefore, objects su
  point numbers, and character str
  elements of a set.
  - The frozenset class is an immutable f
- Python uses curly braces { and }
  - For example, as {17} or {'red', 'green
  - The exception to this rule is that { } d
    set. Instead, the constructor set( ) re

# The dict Class

- Python's **dict** class represents from a set of distinct keys to a
- Python implements a dict using approach to that of a set, but w associated values.
  - The literal form { } produces an em
- A nonempty dictionary is expre separated series of **key:value** dictionary {'ga' : 'Irish', 'de' : 'G 'Irish' and 'de' to 'German'.
- Alternatively, the constructor a key-value pairs as a parameter, pairs = [('ga', 'Irish'), ('de', 'Ge

# Expressions and Op

- Existing values can be
  expressions using spec
  keywords known as op
- The semantics of an op
  upon the type of its op
- For example, when a a
  numbers, the syntax **a**
  addition, while if a and
  operator **+** indicates co

# Logical Operators

- Python supports the fo
  operators for Boolean v

| | |
|---|---|
| **not** | unary negati |
| **and** | conditional |
| **or** | conditional |

- The and and or operato
  in that they do not eva
  operand if the result ca
  based on the value of t

# Equality Operators

- Python supports the fo
to test two notions of e

| | |
|---|---|
| **is** | same ident |
| **is not** | different id |
| == | equivalent |
| != | not equival |

- The expression, a is b,
True, precisely when id
are aliases for the sam
- The expression a == b
general notion of equiv

# Comparison Operat

- Data types may define via the following operat

| | |
|---|---|
| < | less than |
| <= | less than or equ |
| > | greater than |
| >= | greater than or |

- These operators have e for numeric types, and lexicographically, and c strings.

# Arithmetic Operato

- Python supports the following ar

| | |
|---|---|
| + | addition |
| − | subtraction |
| * | multiplicatio |
| / | true division |
| // | integer divisi |
| % | the modulo o |

- For addition, subtraction, and m operands have type int, then the both operands have type float, t
- True division is always of type fl always int (with the result trunca

# Bitwise Operators

- Python provides the following operators for integers:

| | |
|---|---|
| ~ | bitwise complement (pr |
| & | bitwise and |
| \| | bitwise or |
| ^ | bitwise exclusive-or |
| << | shift bits left, filling in v |
| >> | shift bits right, filling in |

# Sequence Operator

- Each of Python's built-in (str, tuple, and list) supp operator syntaxes:

| | |
|---|---|
| s[j] | element at index *j* |
| s[start:stop] | slice including indic |
| s[start:stop:step] | slice including indic |
| | start + 2*step, . . . |
| s + t | concatenation of se |
| k * s | shorthand for s + s |
| val **in** s | containment check |
| val **not in** s | non-containment ch |

# Sequence Comparis

- Sequences define compari
  on lexicographic order, per
  by element comparison un
  is found.
  - For example, **[5, 6, 9]** < **[5,**
    at index 1.

| | |
|---|---|
| s == t | equivalent (element |
| s != t | not equivalent |
| s < t | lexicographically les |
| s <= t | lexicographically les |
| s > t | lexicographically gre |
| s >= t | lexicographically gre |

# Operators for Sets

- Sets and frozensets support the following operators:

| | |
|---|---|
| key **in** s | containment check |
| key **not in** s | non-containment ch |
| s1 == s2 | s1 is equivalent to s |
| s1 != s2 | s1 is not equivalent |
| s1 <= s2 | s1 is subset of s2 |
| s1 < s2 | s1 is proper subset o |
| s1 >= s2 | s1 is superset of s2 |
| s1 > s2 | s1 is proper superse |
| s1 \| s2 | the union of s1 and |
| s1 & s2 | the intersection of s |
| s1 – s2 | the set of elements i |
| s1 ^ s2 | the set of elements i |

# Operators for Dictio

- The supported operato type dict are as follows

| | |
|---|---|
| d[key] | value associated with |
| d[key] = value | set (or reset) the value |
| **del** d[key] | remove key and its as |
| key **in** d | containment check |
| key **not in** d | non-containment chec |
| d1 == d2 | d1 is equivalent to d2 |
| d1 != d2 | d1 is not equivalent to |

# Operator Precedence

| Operator Precedence | | |
|---|---|---|
| | **Type** | **Symbols** |
| 1 | member access | expr.member |
| 2 | function/method calls<br>container subscripts/slices | expr(...)<br>expr[...] |
| 3 | exponentiation | ** |
| 4 | unary operators | +expr, −exp |
| 5 | multiplication, division | *, /, //, % |
| 6 | addition, subtraction | +, − |
| 7 | bitwise shifting | <<, >> |
| 8 | bitwise-and | & |
| 9 | bitwise-xor | ^ |
| 10 | bitwise-or | \| |
| 11 | comparisons<br>containment | **is**, **is not**, =<br>**in**, **not in** |
| 12 | logical-not | **not** expr |
| 13 | logical-and | **and** |
| 14 | logical-or | **or** |
| 15 | conditional | val1 **if** cond **e** |
| 16 | assignments | =, +=, −= |

# Program Structure

- Common to all control structures
  is used to delimit the beginning
  acts as a body for a control struc
- If the body can be stated as a si
  statement, it can technically plac
  to the right of the colon.
- However, a body is more typicall
  indented block starting on the lin
- Python relies on the indentation
  extent of that block of code, or a
  code within.

# Conditionals

```
if first_condition:
    first_body
elif second_cond:
    second_body
elif third_conditi
    third_body
else:
    fourth_body
```

# Loops

- While loop:

```
while condition:
    body
```

- For loop:

```
for element in iterable:
    body                    # body may r
```

- Indexed For loop:

```
big
for
if
```

# Break and Continue

- Python supports a **break** stat
immediately terminate a while
executed within its body.

```
found = False
for item in data:
    if item == targ
        found = True
        break
```

- Python also supports a **conti**
causes the current iteration o
but with subsequent passes o
as expected.

# Functions

- Functions are defined using the ke

```
def count(data, targ
    n = 0
    for item in data:
        if item == targe
            n += 1
    return n
```

- This establishes a new identifier as (count, in this example), and it est parameters that it expects, which **signature**.
- The **return** statement returns the and terminates its processing.

# Information Passing

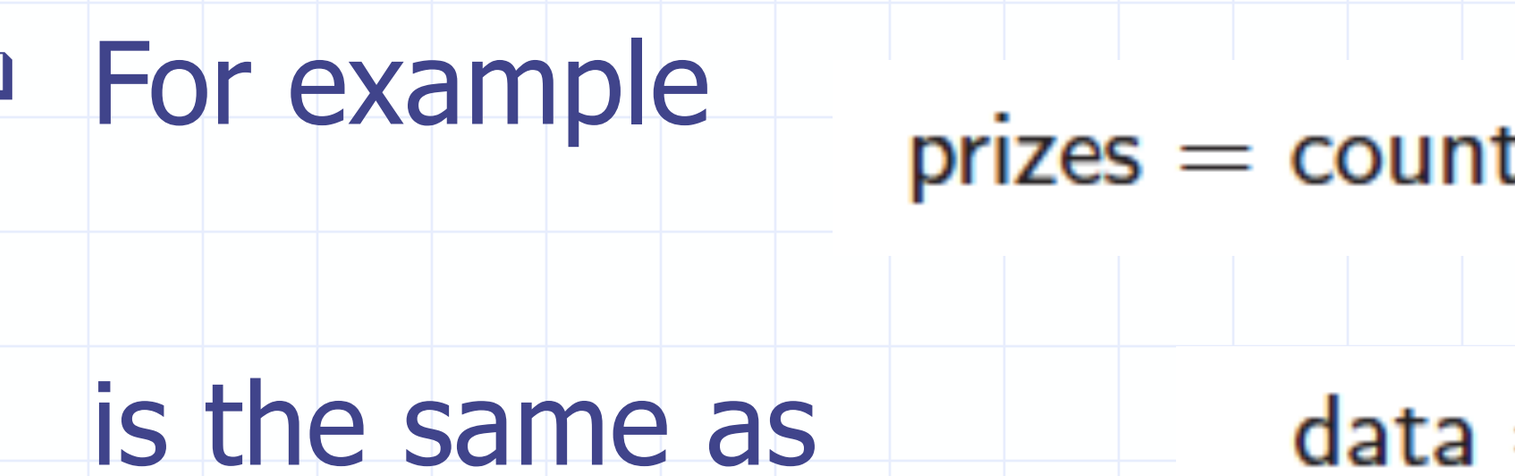
- Parameter passing in Python fo
  the standard assignment state
- For example

```
prizes = count(gr
```

  is the same as

```
data =
target =
```

  and results in



# Simple Output

- The built-in function, **print**, can generate standard output
- In its simplest form, it prints a sequence of arguments, separated and followed by a trailing newline
- For example, the command print('maroon') outputs the string 'maroon'
- A nonstring argument x will be converted to str(x).

# Simple Input

- The primary means for acquiri
  user console is a built-in functi
- This function displays a prompt
  optional parameter, and then w
  enters some sequence of chara
  return key.
- The return value of the functio
  characters that were entered s
  key.
  - Such a string can immediately be c

```
year = int(input('In what ye
```

# A Simple Program

- Here is a simple progra
  some input and output

```
age = int(input('Enter your age in ye
max_heart_rate = 206.9 - (0.67 * age)
target = 0.65 * max_heart_rate
print('Your target fat-burning heart
```

# Week 1 Reading List

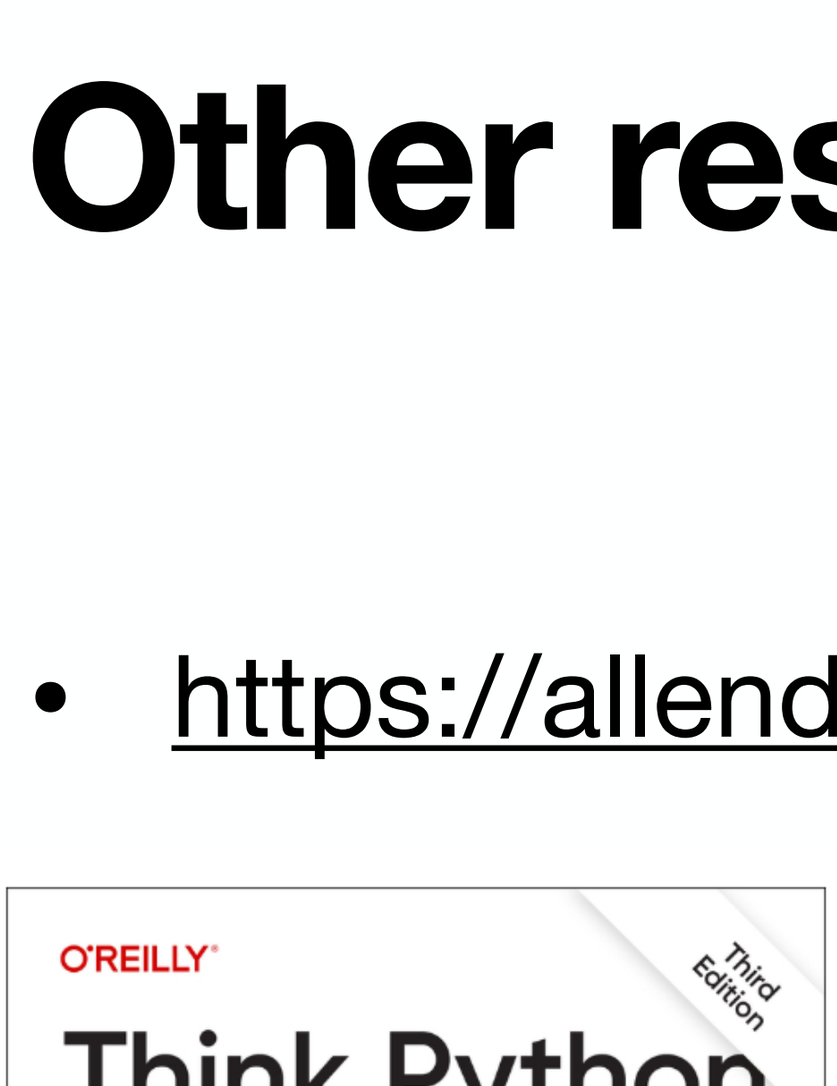
- [1] Chapter 1
- [2] Chapter 1

[1] Problem Solving with Algorithms and Data S
Miller and David Ranum, Luther College, freely

[2] Data Structures and Algorithms in Python,
by Michael T. Goodrich, Roberto Tamassia, Mic
SJSU library via link [2]

# Other resources

- https://allendowney.github.io/ThinkPython/



# Week 2 Reading List

- [1] Chapter 2, 3, 4
- [2] Chapter 2, 4

[1] Problem Solving with Algorithms and Data ...
Brad Miller and David Ranum, Luther College,
via link [1]

[2] Data Structures and Algorithms in Python,
by Michael T. Goodrich, Roberto Tamassia, Mi
via SJSU library via link [2]