جامعة الكويت
KUWAIT UNIVERSITY

CpE Tutorial: Programming with Python

Ameer Mohammed

Computer Engineering Department

Kuwait University

ameermohammed.com/pytutorial

Topics

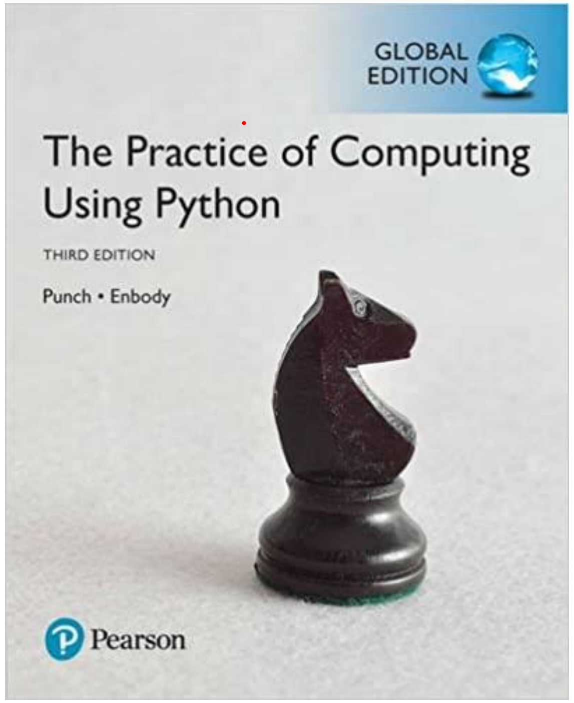
- The Python language and environment
- Variables and Data Types
- Control structures
- Functions
- Classes and Objects
- Other topics: GUI, APIs, Containers, etc. (if time permits)

This box will appear when we want to highlight a difference between **Java** and **Python**

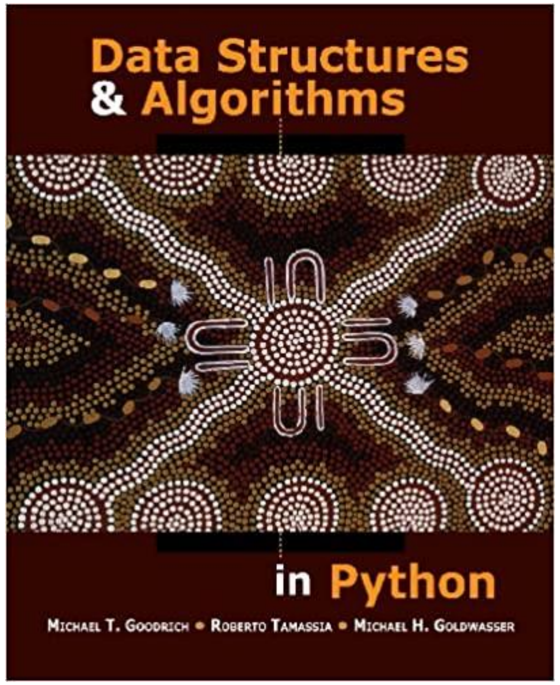
The New Textbooks

The Department Council SharePoint contains:

- Lecture slides
- Instructor material
- Source code
- Figures



CpE 201
Advanced Programming


Data Structures & Algorithms
in Python
Michael T. Goodrich • Roberto Tamassia • Michael H. Goldwasser

CpE 207
Data Structures

The History of Python

Guido van Rossum

Monty Python

- Python 1.0:
  - Initial design by Guido van Rossum
  - Released in January 1994
- Python 2.0:
  - Released in October 2000
  - Introduced list comprehensions, augmented assignments, and string methods
- Python 3.0:
  - Released in December 2008
  - Major backwards-incompatible changes from 2.x
  - Several changes in syntax
  - Current stable version is 3.8.3 (13th May 2020)

Paradigms for Programming Languages

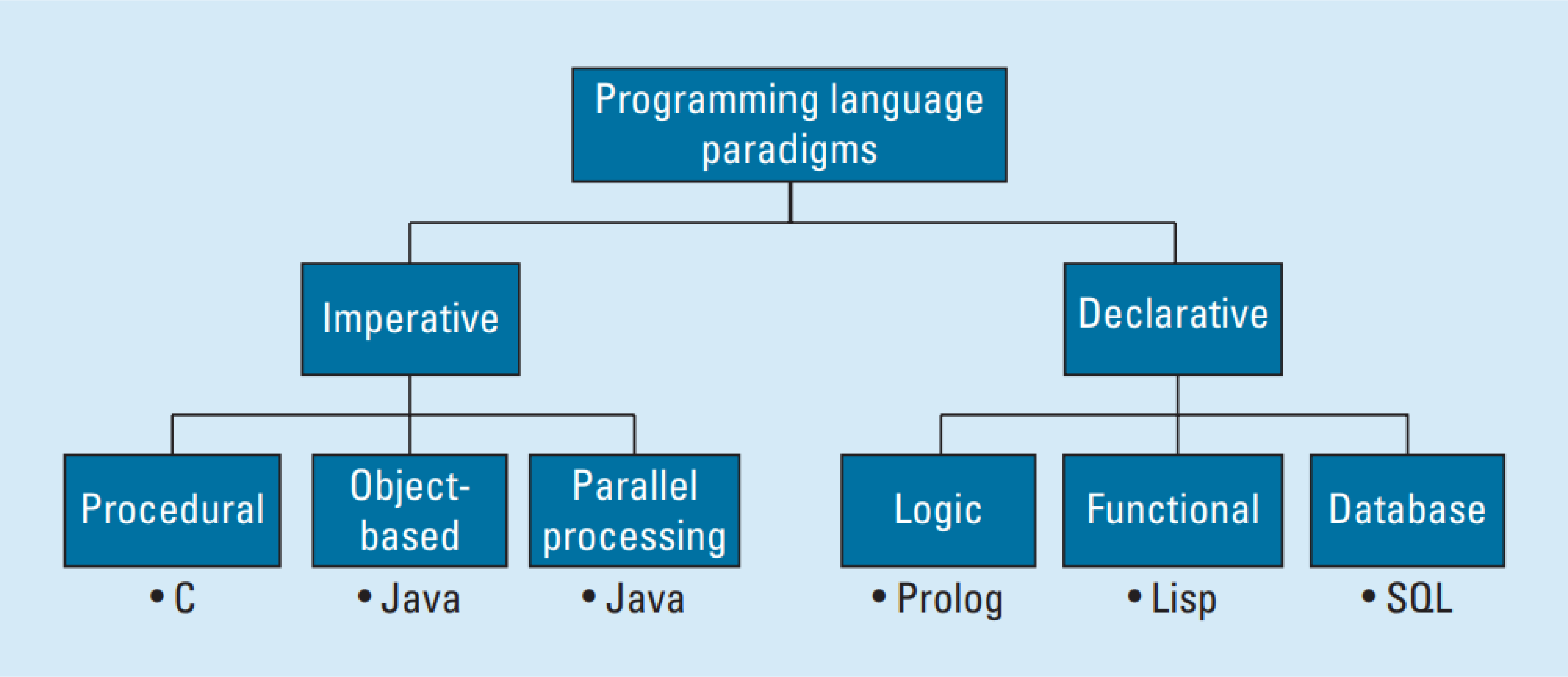
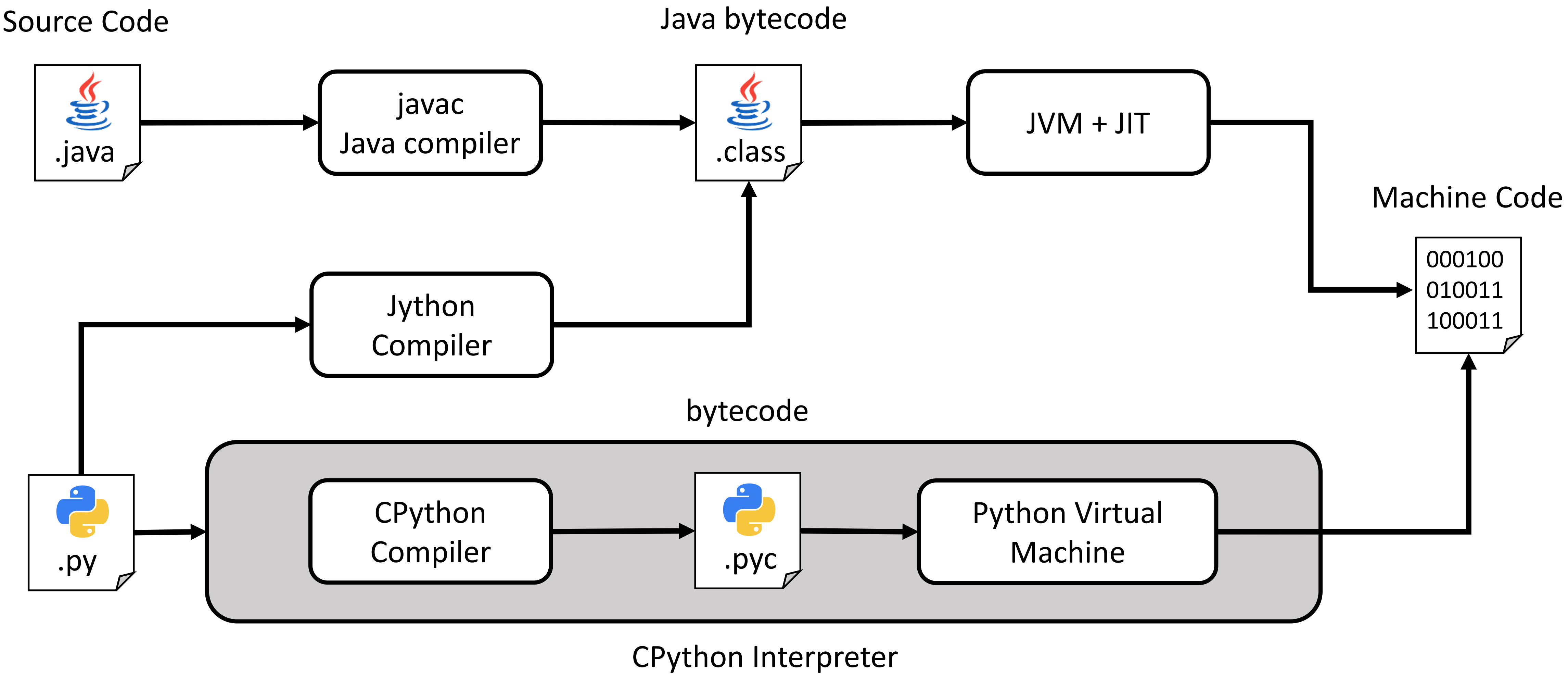


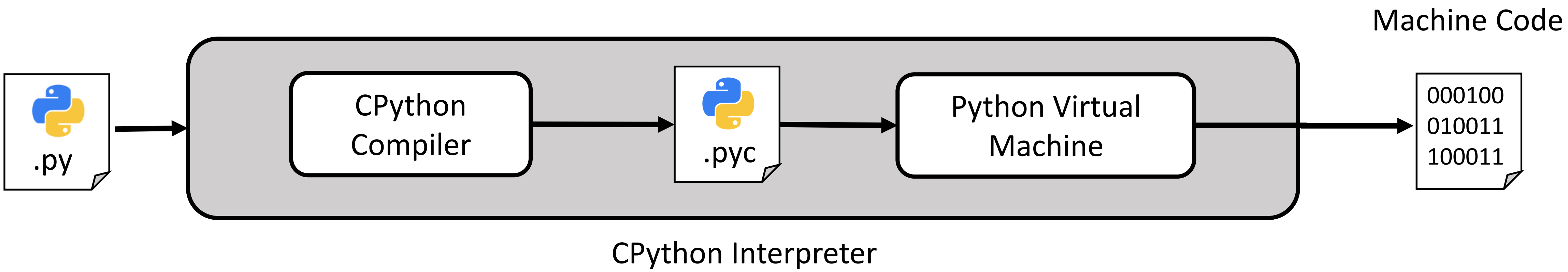
Image courtesy of Laine et al. “Toward Unified Web Application Development” – IT Professional 2011

Python vs. Java Engines

Java is a compiled language
Python is an interpreted language



Python vs. Java Engines



Integrated Development Environment (IDE)

PyCharm Professional	Repl.it
Local cross-platform IDE	Online cloud-based IDE
Local development (with VCS Integration)	Collaborative development
Extensive debugger	Simple debugger
Supports plug-ins and extensions	Facilitates learning with code tools and templates

First Program: Hello World

- Let’s go to repl.it and open up a new Repl

Variables

- A **variable** is a name we designate to represent an object (number, data structure, function, etc.) of a specific **data type** in our program.

- We use names to make our program more readable, so that the object is easily understood in the program.

- Variable names cannot be the same as Python keywords

- Variables are assigned values using the = operator (called the assignment operator)

```
my_int = 3
my_str = “sandwich”
```

Built-in Basic Data Types and Operators

Data Type	Operators (ascending precedence)	Example
integer	+ - % / // * ** ()	7
float		3.142
complex		complex(2,8)
Bool		True
str	+	“Hello”

Java does not have ** Needs Math.exp() for exponentiation

Java numeric types are size-bounded (int, long, double). Python integer sizes are unbounded.

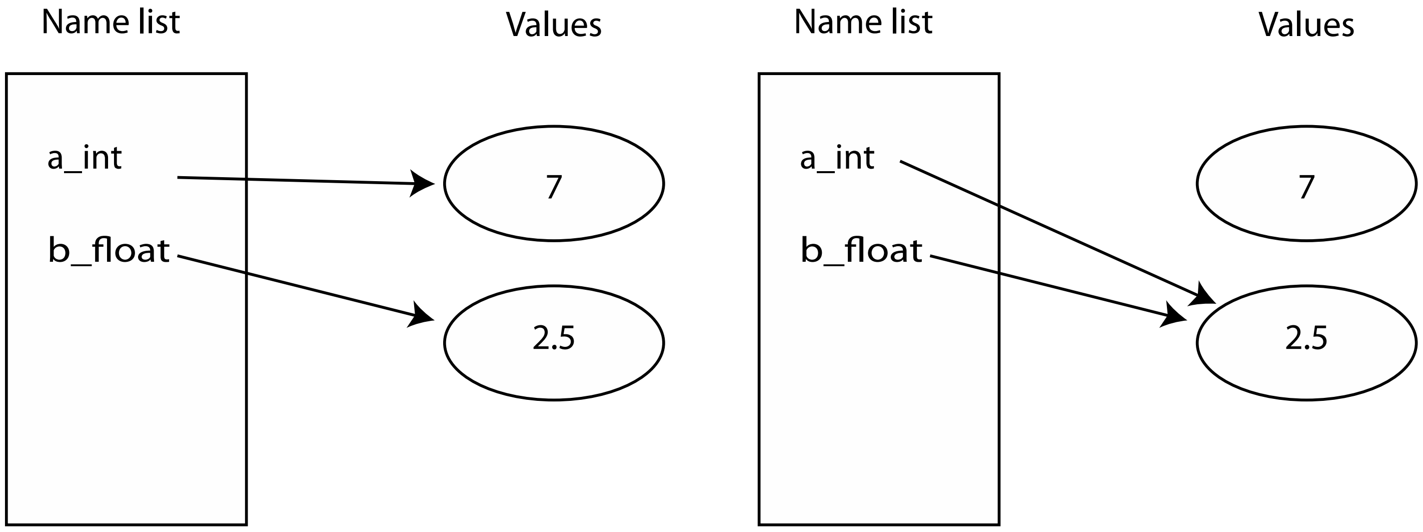
Example: Variables and Data types

- Create variables that hold different data types then manipulate these variables using various operators.

Variables

- **Namespace** contains all variables currently assigned values in the program.

- **Datatype** of variables change depending on what value they reference.



```
a_int = 7
b_float = 2.5
```

```
a_int = b_float
```

Example: Volume of a cylinder

- Write a program that accepts as input the height and radius of the cross-section of cylinder, then outputs the result in the console.

Importing modules

- A **module** is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

- In order to use the statements of a specific module within your program, you need to **import** it.

- To import the entire module: **import** math

- To import a specific definition from a module: **from** math **import** pi

Java vs. Python: Data Types and Operators

1. In Python, all data types are considered **objects**
   - They have attributes and methods

2. In Python, variables do not need to be declared
   - They are automatically defined as soon as they are assigned a value
   - They are **dynamically-typed**: their type can change at runtime
   - Explicit type conversion is possible

Java has "primitive" data types occupying fixed memory (e.g. int)

Java is statically typed: you need to declare a variable with its type at compile time

Setting up the development environment

ameermohammed.com/pytutorial

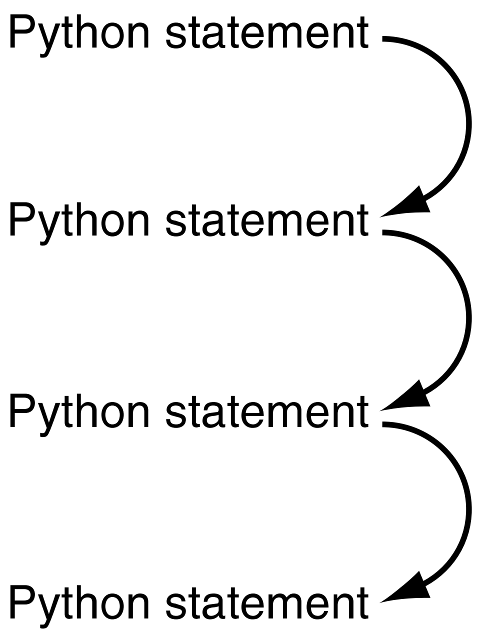
Built-in Collection Data Types

Collection Type	Properties	Mutable?	Example	Java Counterpart
List	Ordered indexed collection	Yes	[1, 2, ‘a’, 8.6, 1]	ArrayList
Dictionary	Hash table of key-value pairs, non-indexed	Yes	{1:’a’, 2:’b’, 3:’c’}	HashMap
Set	Unordered non-indexed list, no duplicates	Yes	{‘h’, 3, ‘k’}	HashSet
Tuple	Immutable list	No	(1, 2, ‘a’, 8.6, 1)	-
Range	Immutable, homogeneous list	No	range(0,10)	-

Topics

- The Python language and environment
- Variables and Data Types
- Control structures
- Functions
- Classes and Objects
- Other topics: GUI, APIs, Containers, etc. (if time permits)

Sequential Programs



Python statement

Python statement

Python statement

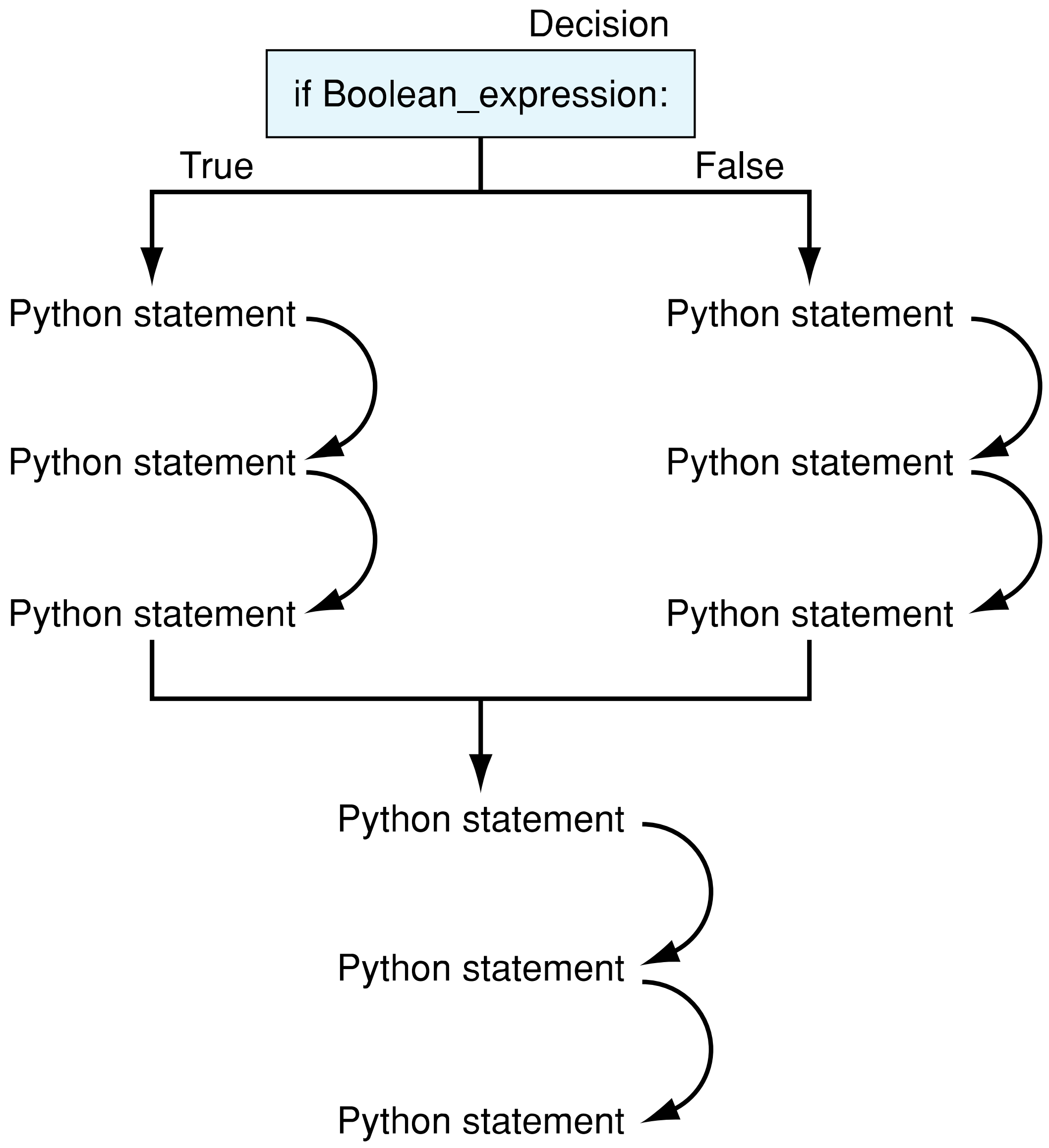
Python statement

Selection Statements

```
if boolean_expression:
    statement1
    statement2
else:
    statement3
    statement4
```

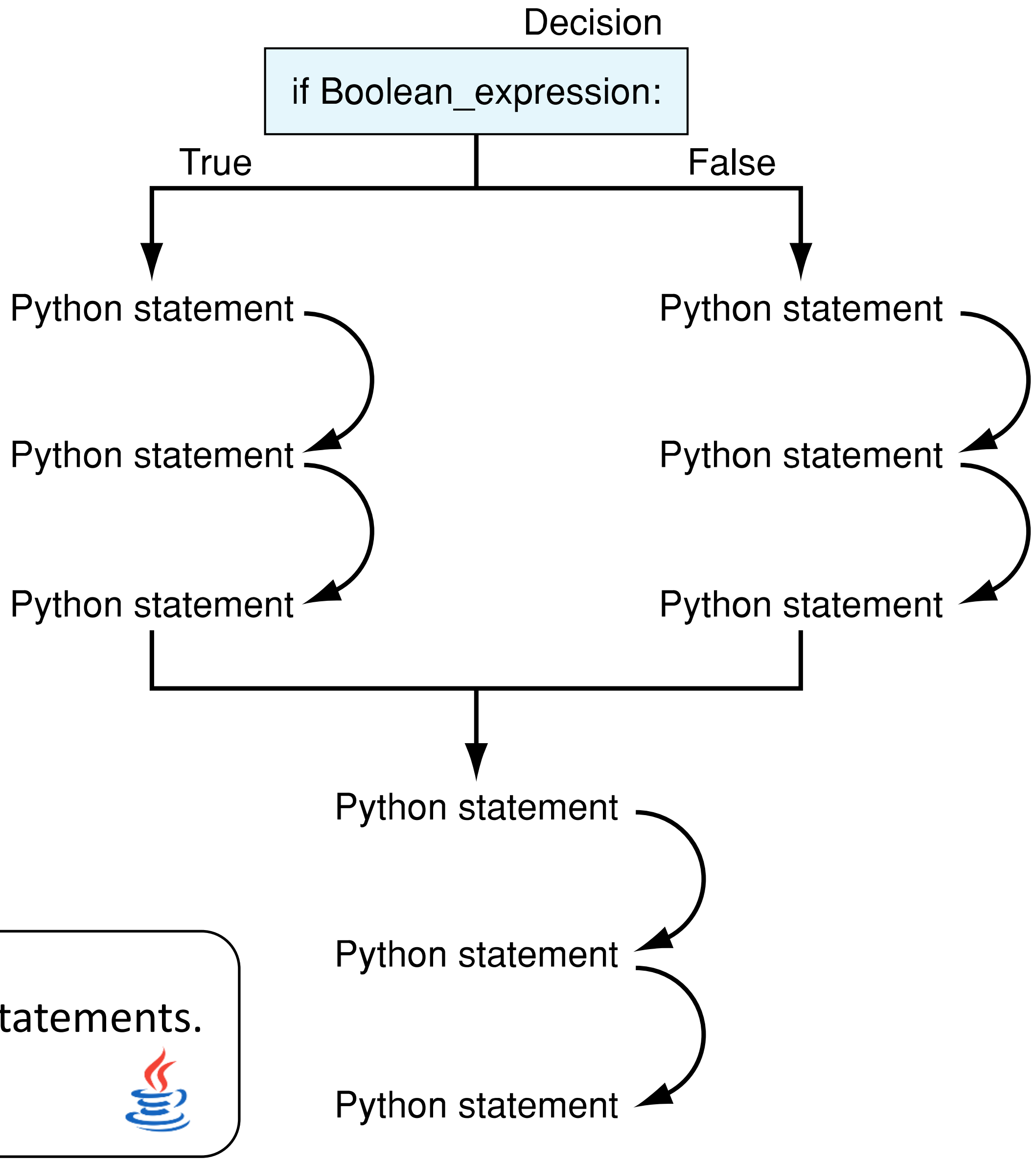
indent →

Unlike Python, **Java** encloses a block of statements in { } instead of indentation



Selection Statements

```
if boolean_expression:
    statement1
    statement2
elif:
    statement3
    statement4
else:
    statement5
    statement6
```



Java also has switch statements.
Python does not.

Selection Statements

- Boolean Expressions
  - These are expressions that evaluate to True or False. They are composed of variables/expressions as operands acted on by relational/logical operations.

Relational Operator
<
<=
>
>=
==
!=
is, is not
in, not in

Logical Operator
and
or
not

Java uses &&, ||, and ! for its logical operators

Selection Statements

- Examples: Boolean Expressions

- a == 3
- y > 50
- 10 <= x <= 20
- x > 20 and (y < 50 or z > 30)
- “apple” in my_list

Example: Login Module

- Write a program that takes as input a name and ID then outputs the following to the console:

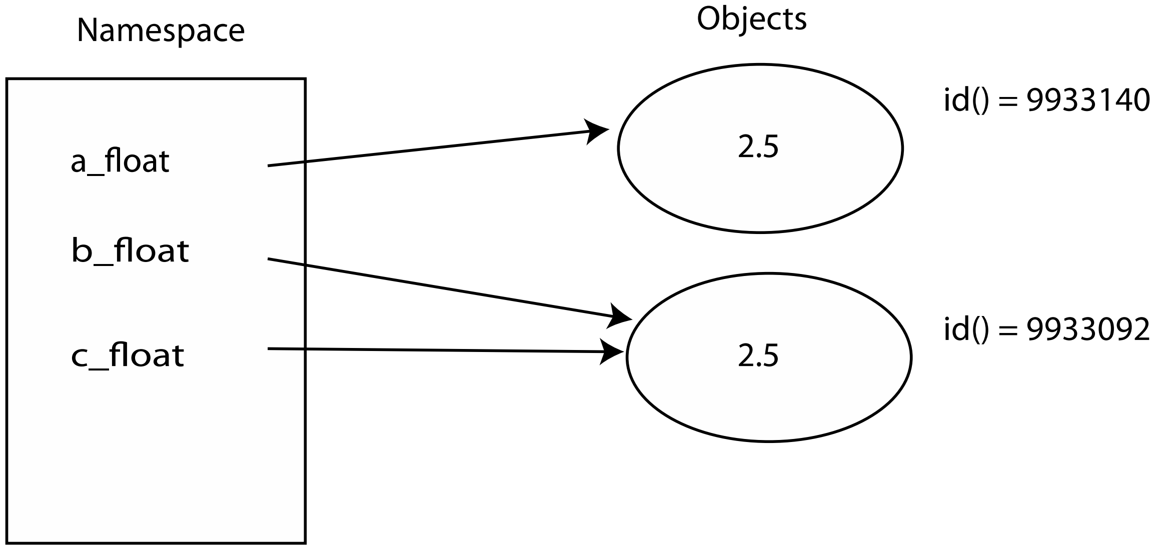
“Welcome to the system {NAME}. You are a {ROLE}”

- Where {NAME} is replaced with the entered name
- If the ID is between 200 and 250 then the system should instead output “You are denied access to the system”.
- {ROLE} is substituted with “manager” if ID is less than 100 otherwise it is substituted with “employee”

Selection Statements

- Boolean Expressions
    - Difference between “==“ and “is” operators

```
a_float = 2.5
b_float = 2.5
c_float = b_float
```

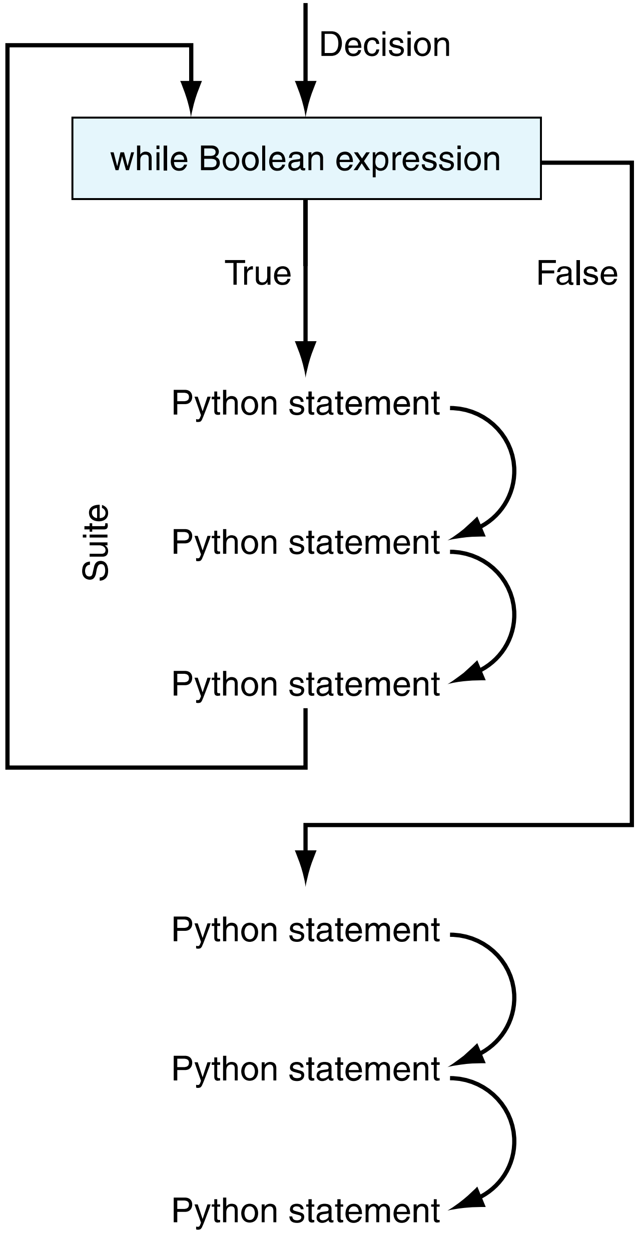


```
a_float == b_float → True
a_float is b_float → False
b_float is c_float → True
```

Repetition Statements

while *boolean_expression*:
    statement1
    statement2

statement3
statement4

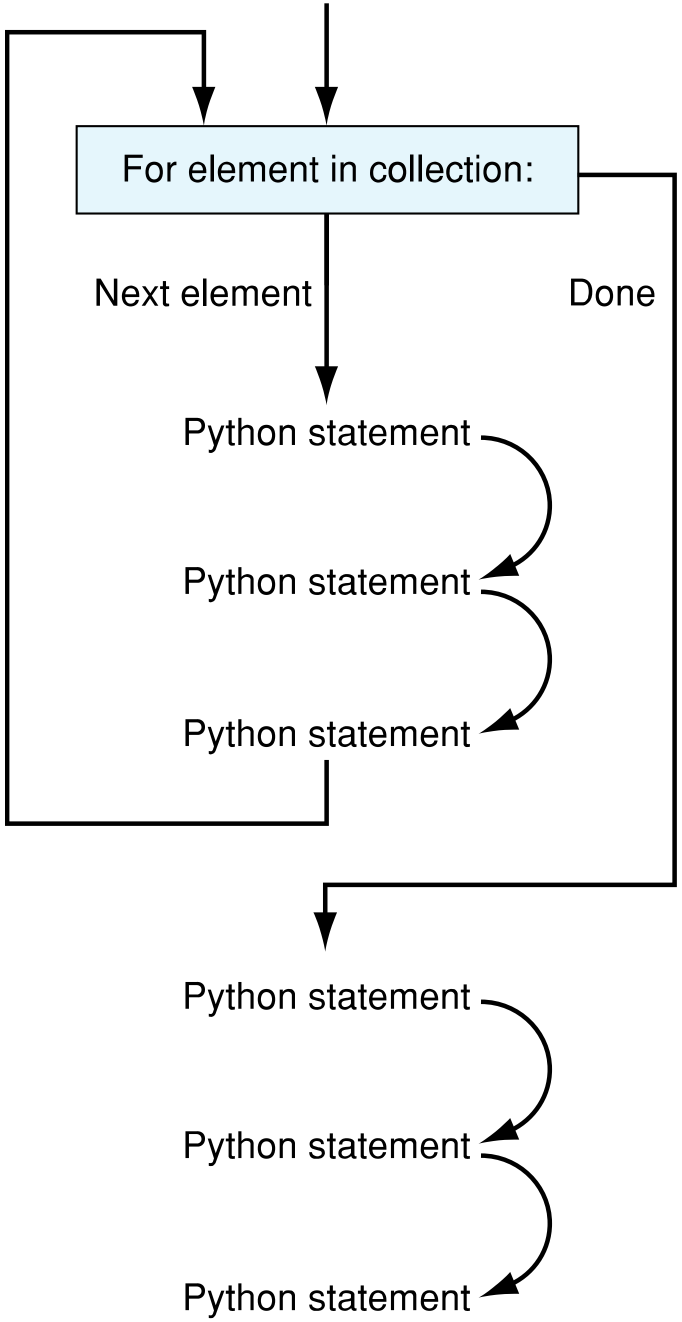


Example: Printing Even Numbers

- Write a program that prints out all even numbers from 10 to 40 then outputs their sum.

Repetition Statements

```
for element in collection:
    statement1
    statement2
    statement3
```



Example: Newsletter

- Create a list of customer names then print the following message on the console for each customer on your list:

“Thank you for subscribing, {NAME}”

- Where {NAME} is replaced with the customer name.

Example: Printing Even Numbers

- Write a program that prints out all even numbers from 10 to 40 then outputs their sum.

- Use the **for** control structure this time over the **range** collection.

Topics

- The Python language and environment
- Variables and Data Types
- Control structures
- Functions
- Classes and Objects
- Other topics: GUI, APIs, Containers, etc. (if time permits)

Reusing Code

- Suppose you wrote some code that you would like to use in several places in your program.

- Example: Computing the volume of a cylinder

```
vol = math.pi * (rad_int ** 2) * height_int
#....some code
vol2 = math.pi * (rad_int2 ** 2) * height_int2
```

Not easy to maintain or share

Functions

- **Functions** are segments of code that perform some operation and return one value.
- They "encapsulate" the performance of some particular operation, so it can be used by others (for example, the sqrt() function)
- Once defined, functions can be **called** (or invoked) by other sections of the program.
- They are an abstraction of an operation that facilitates:
    - Modularity
    - Reusability
    - Security
    - Maintainability

Using Functions

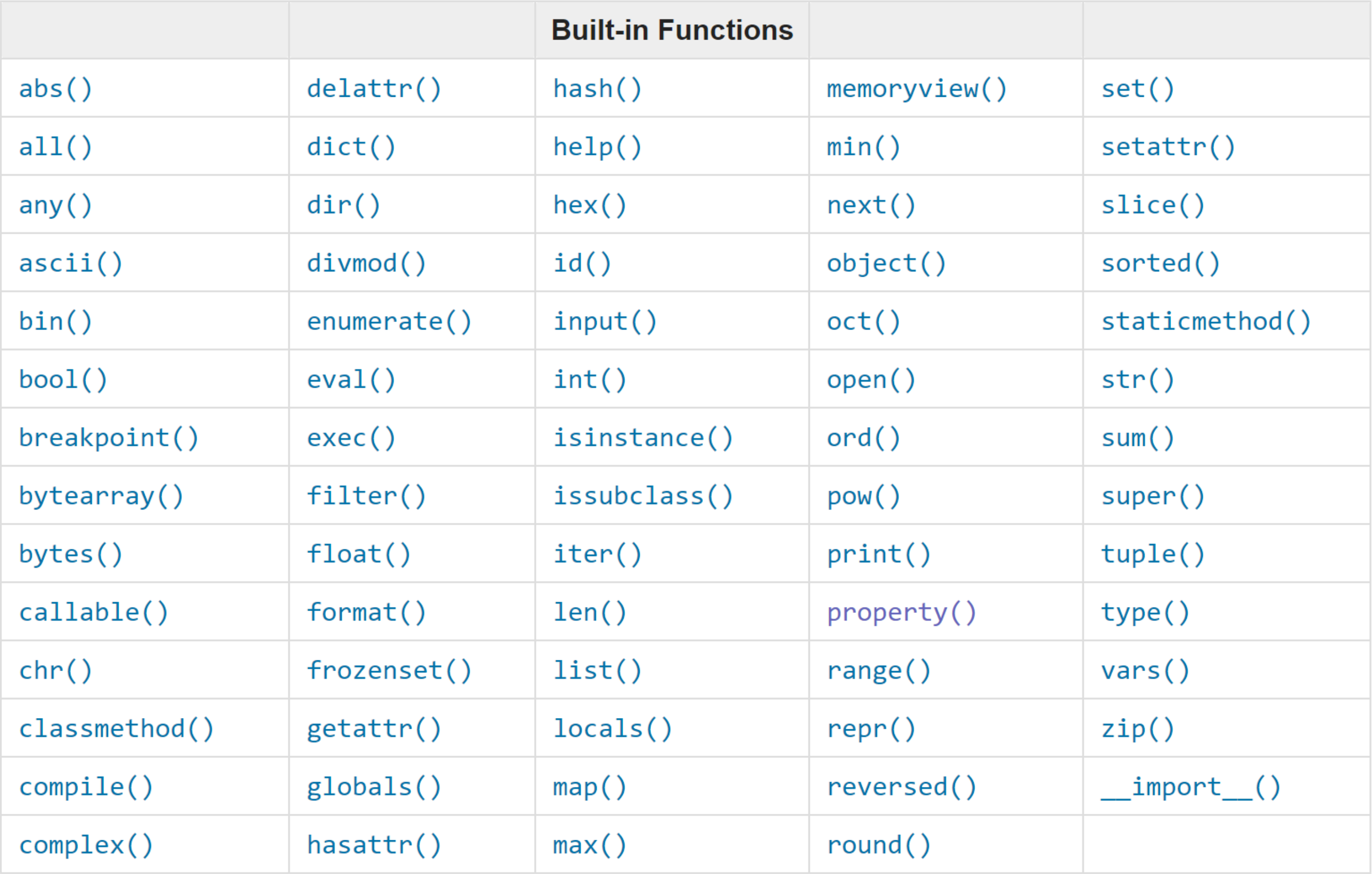
- Example: Computing the volume of a cylinder

```
vol = calc_vol(r, h)
#....some code
vol2 = calc_vol(r2, h2)
```

Easy to maintain and share

Built-in Functions

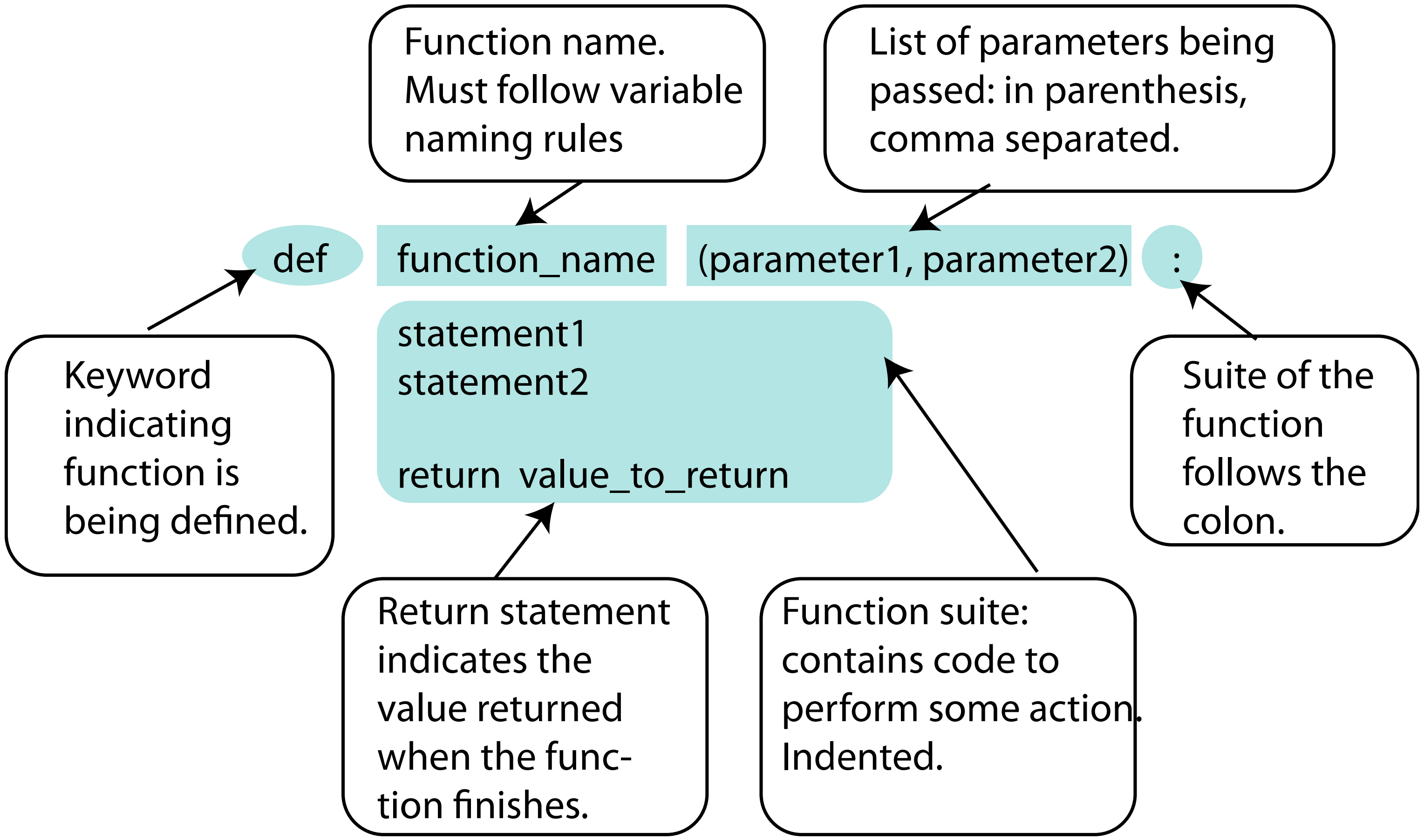
- The Python interpreter has a number of functions and types built into it that are always available.



		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

https://docs.python.org/3/library/functions.html

Defining your own functions



Function name. Must follow variable naming rules

List of parameters being passed: in parenthesis, comma separated.

```
def function_name (parameter1, parameter2) :
statement1
statement2

return value_to_return
```

Keyword indicating function is being defined.

Suite of the function follows the colon.

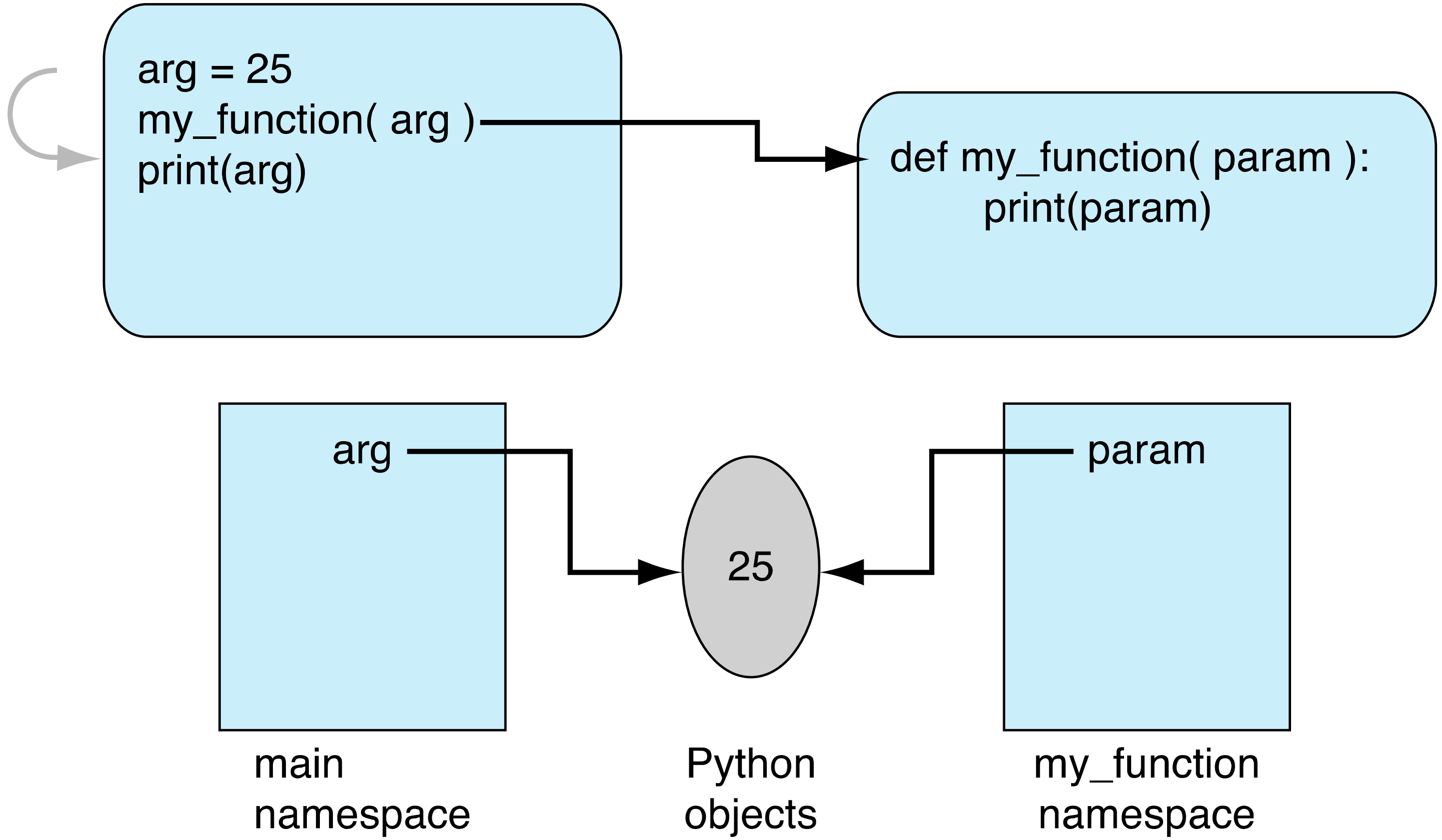
Return statement indicates the value returned when the function finishes.

Function suite: contains code to perform some action. Indented.

Example: Volume of a Cylinder

- Create a function that takes as input the radius and height of the cylinder then returns the volume.

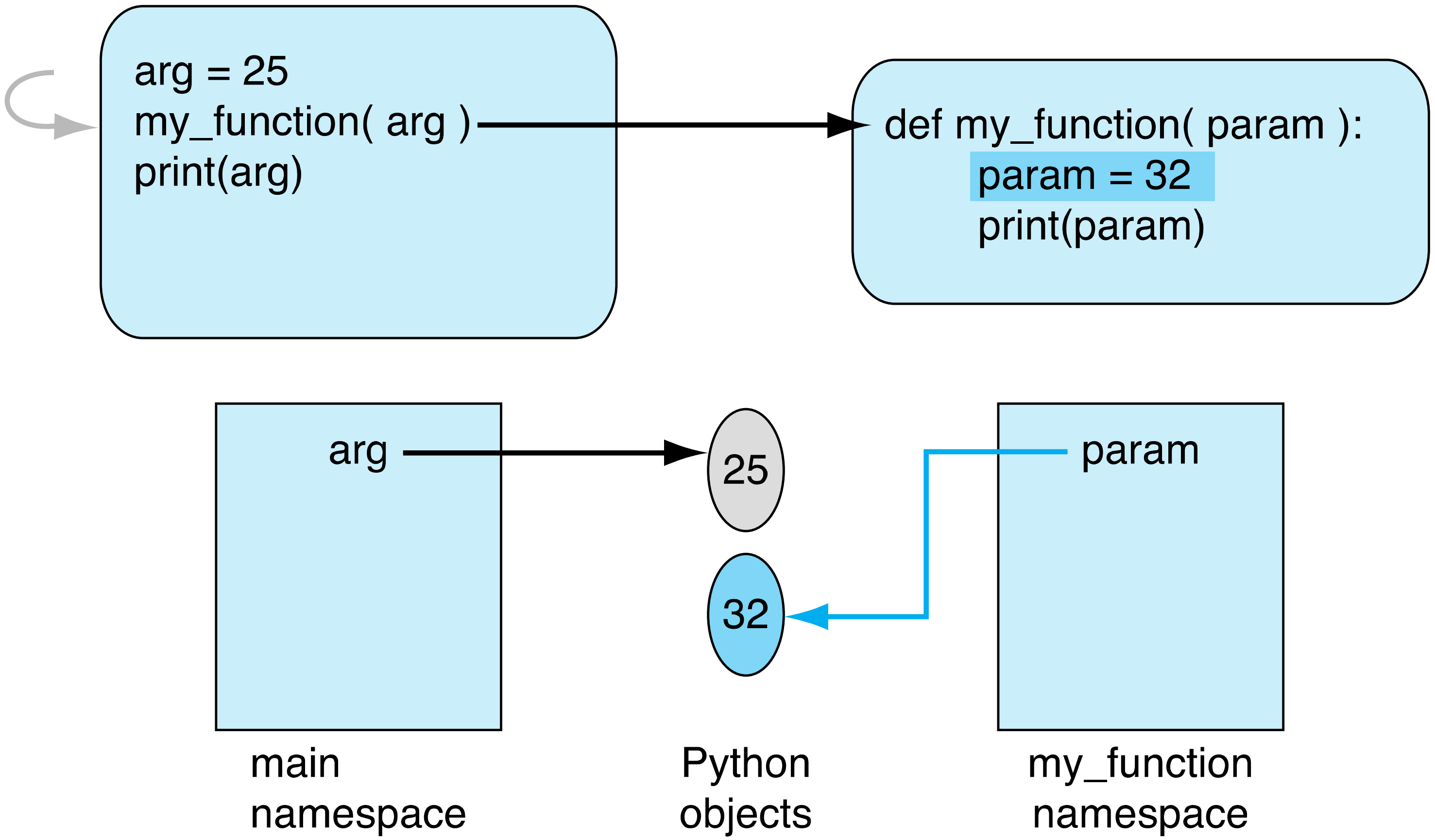
Variable Scope and Function Parameters



```
arg = 25
my_function( arg )
print(arg)
```

```
def my_function( param ):
        print(param)
```

Passing Immutable Objects

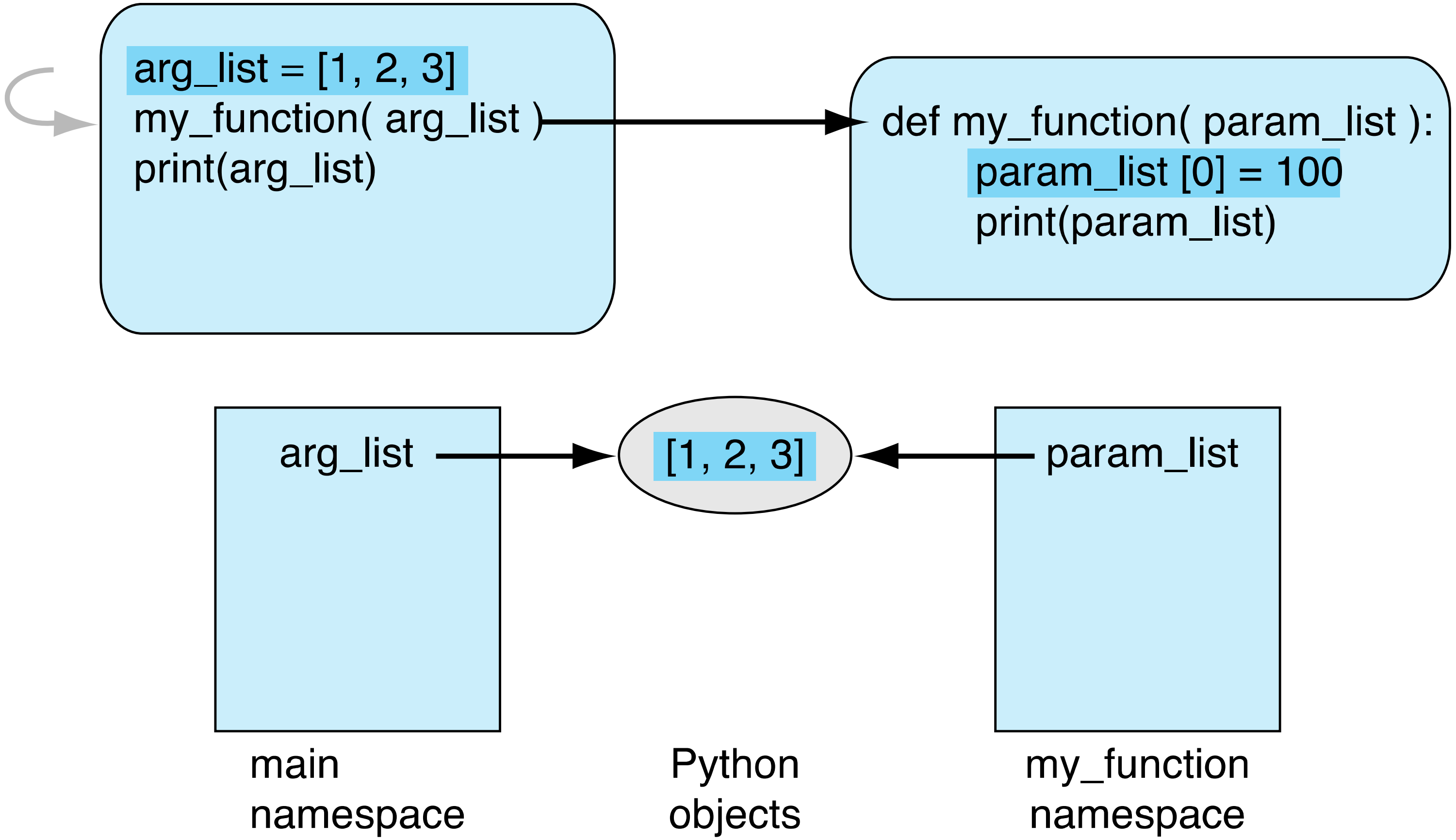


```
arg = 25
my_function( arg )
print(arg)
```

```
def my_function( param ):
        param = 32
        print(param)
```

Immutable Objects
Integer
Float
Bool
String
Tuple
Range

Passing Mutable Objects



```
arg_list = [1, 2, 3]
my_function( arg_list )
print(arg_list)
```

```
def my_function( param_list ):
        param_list [0] = 100
        print(param_list)
```

Mutable Objects
List
Dictionary
Set

Pass-by-value or pass-by-reference??

Default and Named Parameters

```
def box(height, width=10,depth=10, color= "blue" ):
    print(height, width, length)
    ...other statements
```

The parameter assignment means two things:

- **Defaults**: If the caller does not provide a value, the default is the parameter assigned value
- **Named**: You can get around the order of parameters by using the name.

```
box(10, depth=4)
```

Example: Student Grade Manager

- Create a function that takes as input a student grades and a list of weights then outputs the student’s final numeric grade.
- Create a function that takes as input the student’s numeric grade and outputs the letter grade.

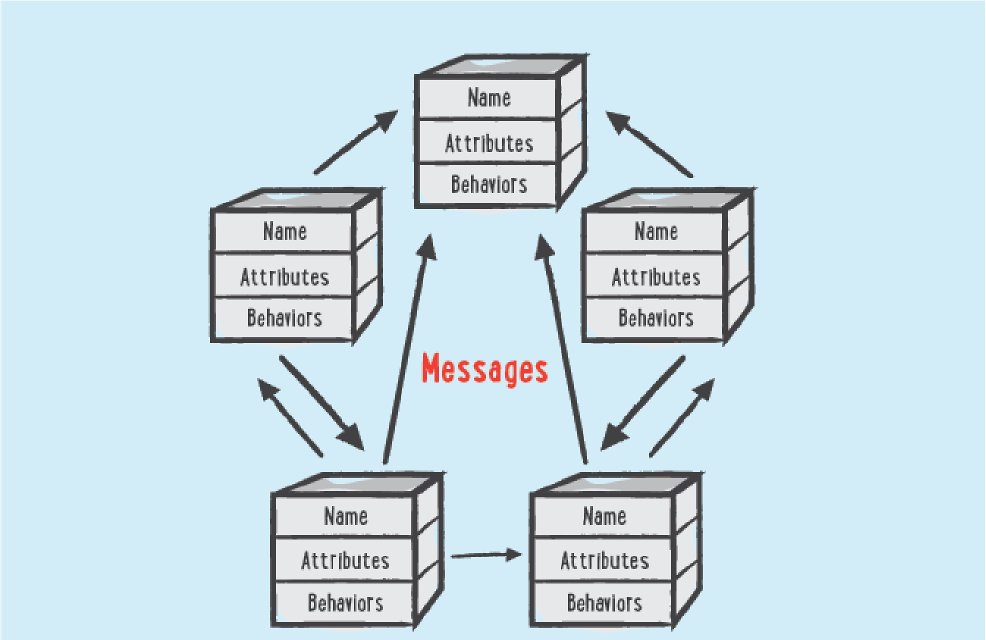
- Write a program that uses these functions to compute the letter grades of any given student.

Topics

- The Python language and environment
- Variables and Data Types
- Control structures
- Functions
- Classes and Objects
- Other topics: GUI, APIs, Containers, etc. (if time permits)

Object-Oriented Programming (OOP)

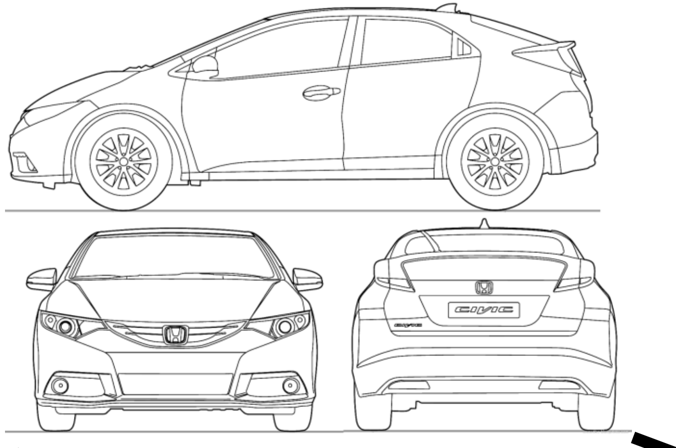
- Object oriented programming is a way to think about "objects" in a program (such as variables, functions, etc.)
- A program becomes less a list of instruction and more a set of objects and how they interact.
- A **class** in a program represents a user-defined data type that one can use to create objects (or **instances**) of the same structure defined by the class.



An object-orientied program consists of many well-encapsulated objects and interacting with each other by sending messages

ILLUSTRATED BY SEGUE TECHNOLOGIES

Class and Object



Civic Class:

- Color
- Suspension
- Tires
- Engine
- ….

Object

Instance 1:
- Color: white
- Engine: …

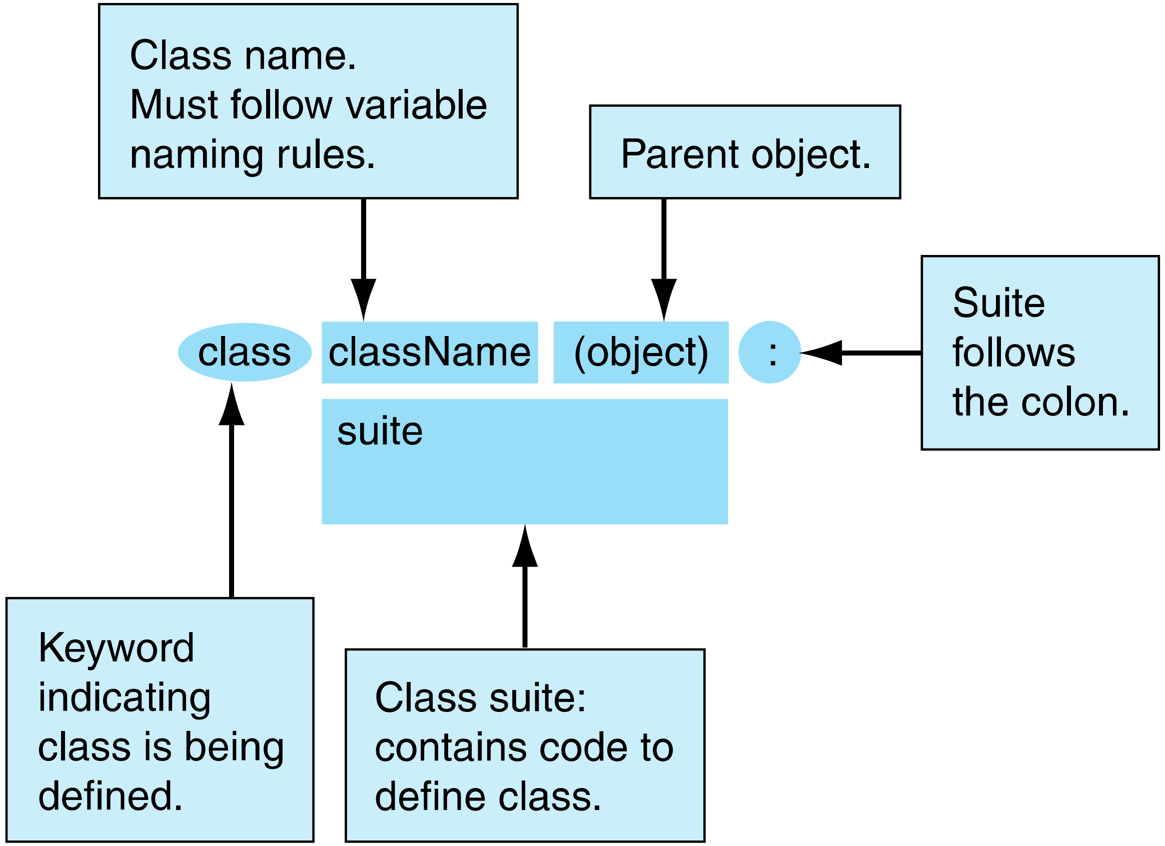
Object

Instance 2:
- Color: green
- Engine: …

Instance 3:
- Color: blue
- Engine: …

Object

Defining new classes



Class name. Must follow variable naming rules.

Parent object.

Suite follows the colon.

Keyword indicating class is being defined.

Class suite: contains code to define class.

Components of a Class Definition

- **Constructor**: used to initialize the data attributes of a new instance.

- **Class Attributes**: consist of
  - **Class-wide attributes:** shared by all instances.
  - **Methods**: functions that "act" on an instance

Static class variables

- **Data Attributes**: instance-specific variables

Fields

Example: Student Class

- Create a new class representing a student data type with attributes: first name, last name, and ID. Generate a new instance of Student.

- Instantiate an object of type Student

- Use the dir() built-in function to learn more about your class.

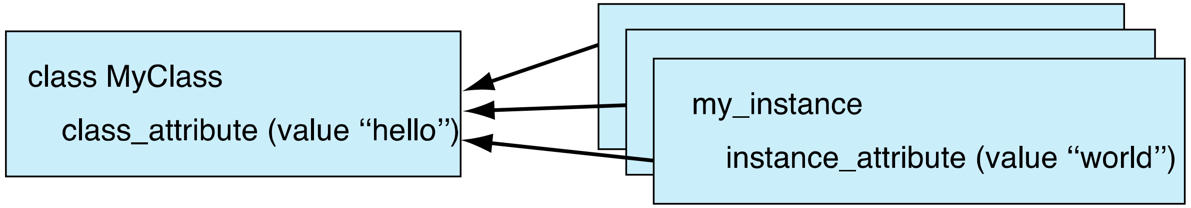
- Use instance_of() on the created object

Instance Creation and Attribute Access

- We can refer to the attributes of an object using “dot” reference, of the form:

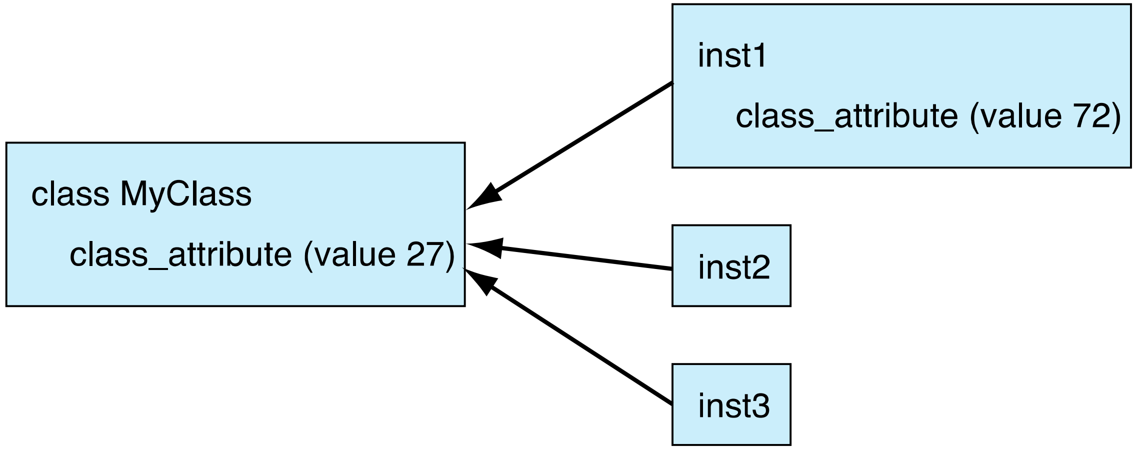
```
my_instance.instance_attribute
```

- The attribute can be a variable or a function (method)
- The attribute is part of the object, either directly or by that object being part of a class.


class MyClass
class_attribute (value “hello”)
my_instance
instance_attribute (value “world”)

Instance Creation and Attribute Access

- Attribute scope is observed between classes and instances.
  - Classes can only access class attributes
  - Instances can access class and instance attributes
  - If a class attribute and instance attribute have the same name, instance access takes precedence.

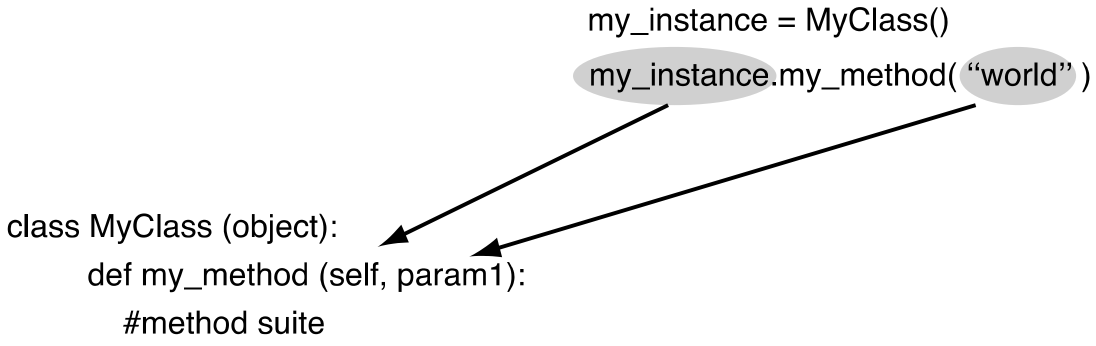


```
inst1 = MyClass()
inst2 = MyClass()
inst3 = MyClass()

MyClass.class_attribute = 27
Inst1.class_attribute = 72
```

Instance Method Call

- Methods are functions defined ***inside*** the suite of a class that behave slightly differently than normal functions when called as an object attribute.
- Methods always bind the first parameter in the definition to the object that called it. The first argument is implicitly passed as the object that called the method.
- This parameter can be named anything, but traditionally it is named ***self***



```
my_instance = MyClass()
my_instance.my_method( "world" )
```

```
class MyClass (object):
    def my_method (self, param1):
        #method suite
```

Instance Private Variables

- Every object has its own namespace to store all its local attributes (variables, functions).
  - Represented by the __dict__ variable

- Can create “private” attributes for instances. These are accessible only within the class and not outside of it.
  - “Private” variables are prefixed with single or double underscore.
  - If double underscore is used, name is mangled: _ClassName__variable

- Note that privacy is **NOT** enforced. If you really want to access it, you still can.

OOP Principles

- ***Encapsulation***: hiding design details to make the program clearer and easier to maintain.
- ***Inheritance***: create a new object by inheriting object characteristics while creating or over-riding for this object
- ***Polymorphism***: allow one message to be sent to any object and have it respond appropriately based on the type of object it is.
    - Python calls this **duck-typing**

Encapsulation

- There is “soft” encapsulation in Python. While it is supported using syntactical conventions, it is not enforced.

- A variable prefixed with an underscore (e.g. _name) **should** be treated by the developer as a **private** part of the object (whether it is a function, a method or a data member).

- Can use decorators to establish “setters” and “getters” for variables. In Python, these methods are called **properties**.

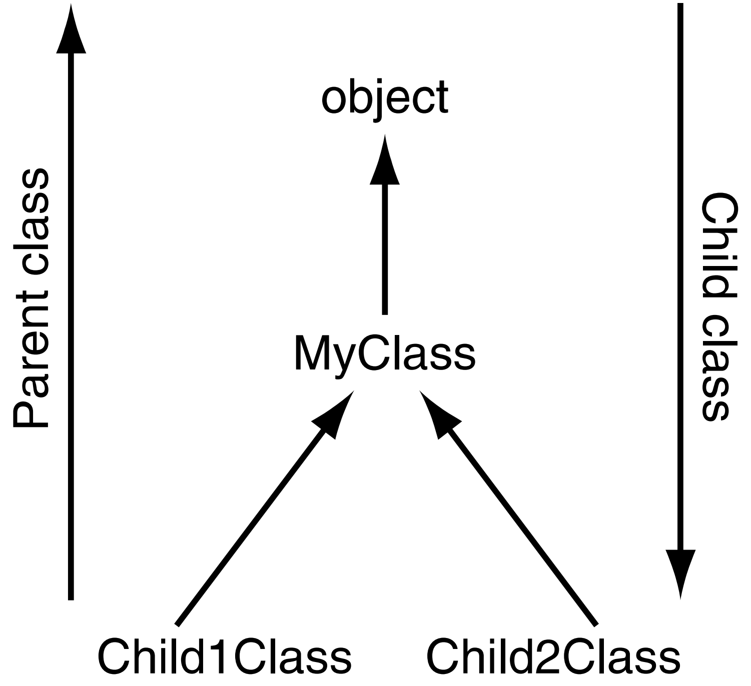
Inheritance

- We can create relationships between classes in such a way so that one class can inherit the structure and behavior of another class.

```
class MyClass (object):
   pass


class Child1Class (MyClass):
   pass


class Child2Class (MyClass):
   pass
```



Why use inheritance?

- ***Specialization:*** A subclass can inherit code from its superclass and anything that is particular to that subclass

- ***Override:*** Change a behavior to be specific to a subclass

- ***Reuse code:*** Use code from other classes (without rewriting) to get behavior in our class.

Example: Inheritance

- Create a new base class called User that should have all the shared logic for any User-type class

- Extend the Student class to inherit from User
  - Update constructor to only create attributes specific to student

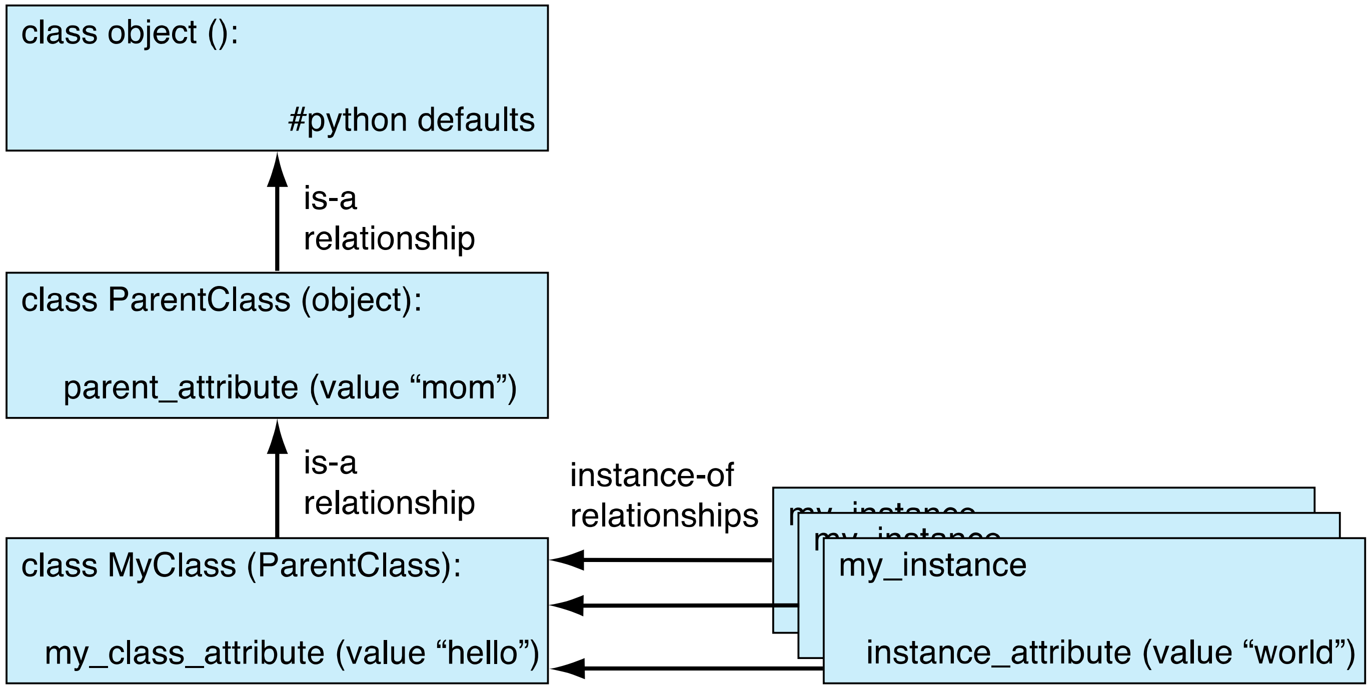
- Create a new class for Faculty that inherits from User

Inheritance

Just like **Java,** methods are virtual by default.

- Attribute Scope

1. Look in the object for the attribute
2. If not in the object, look to the object's class for the attribute
3. If not in the object's class, look up the hierarchy of that class for the attribute
4. If you hit `object`, then the attribute does not exist


class object ():
#python defaults
is-a relationship
class ParentClass (object):
parent_attribute (value “mom”)
is-a relationship
instance-of relationships
class MyClass (ParentClass):
my_class_attribute (value “hello”)
my_instance
my_instance
my_instance
instance_attribute (value “world”)

Inheritance: advanced features

- Python supports multiple inheritance
  - Example: Student can inherit from User and Customer

Java does not have multiple inheritance. Has interfaces instead.

- Python allows you to define **metaclasses**.
  - These define how classes (as objects) are constructed
  - Classes are also objects too and their type is a metaclass called “`type`”
  - Can be used to define classes dynamically (even at runtime!)

Polymorphism

- **Polymorphism:** the behavior of an object changes based on what type it is.
- **Duck-typing**: a form of polymorphism
  - Instead of relying on the type of the object to determine its behavior, the presence of a given method or attribute is checked instead.
- This is realized in Python via:
  - Function overriding: can redefine functions of parent classes
  - Operator overloading: redefining operators for new objects
  - Abstract classes and methods: can be used to force function overriding
- Note: Python does **not** support **function overloading**.

Example: Polymorphism

- Override User methods in Student to specialize it.
- Overload the “+” operator so that adding two students creates a new student with GPA that is the average of the two students’ GPA.
- Overload the “dunder” method __str__ to replace implementation of the built-in function

This is similar to implementing the .toString() method in **Java**

Topics

- The Python language and environment
- Variables and Data Types
- Control structures
- Functions
- Classes and Objects
- Other topics: GUI, APIs, Containers, etc. (if time permits)

Graphical User Interface

- Python has many libraries and frameworks that allow developers to build GUI applications for example:
  - **TkInter**: traditionally bundled with Python as the standard GUI library
  - **PyQT**: cross-platform library implementing the Qt interface
  - **Kivy**: OpenGL ES 2 accelerated framework for the creation of new user interfaces
  - **PySimpleGUI**: a recent third-party library that wraps tkinter to build custom GUI objects

Example: TkInter GUI Library

APIs for Scientific Computing

- There are some very useful libraries available for Python that help in developing extensive computational programs and simulations.
  - NumPy: for linear algebra
  - matplotlib: for plotting and visualization
  - pandas: for data manipulation that resembles relational database operations

- When imported, these libraries offer features that rival that of MATLAB.

Other Topics

- There are other important topics that will be covered in the CpE 201 course which we will not see here:
    - Decorators
    - Files and I/O
    - Exception handling
    - Event-driven programming
    - …

- There are more advanced topics that are very useful to know:
    - Comprehensions
    - Lambda functions (anonymous functions)
    - ctypes