Exercise (from Lectorial 1)

We want to build a simple application that will manage a savings account.

The transactions supported are deposit, withdraw, calculate interest, and check balance.

Problem??

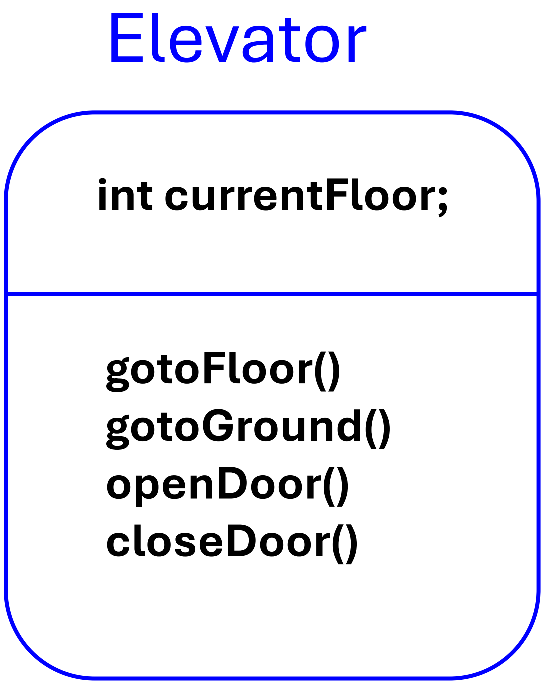
Object?? Class??

Data?? Methods??

How do we use the class to assemble the application?

Test Your Knowledge (from Lectorial 1)

Create a class called **Elevator** based on the diagram below. Write a simple application to simulate the elevator's operation. You may assume that the building has 6 floors.



Elevator

int currentFloor;

gotoFloor()
gotoGround()
openDoor()
closeDoor()

COMP642

Object Oriented Programming

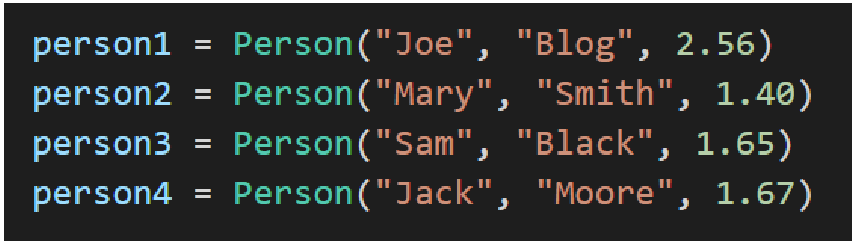
Lectorial 2

Learning Objectives

- Declare and create a list and add objects to it.
- Use list methods and loops to traverse and manipulate the list.
- Understand how lists store references to objects and the difference between value and reference types.
- Check object references and determine if variables refer to the same object or if they refer to an object at all.

Tracking Multiple Objects

- Often, we want to keep track of many objects.
- Examples:
  - All cars in a sales yard
  - All students in a class

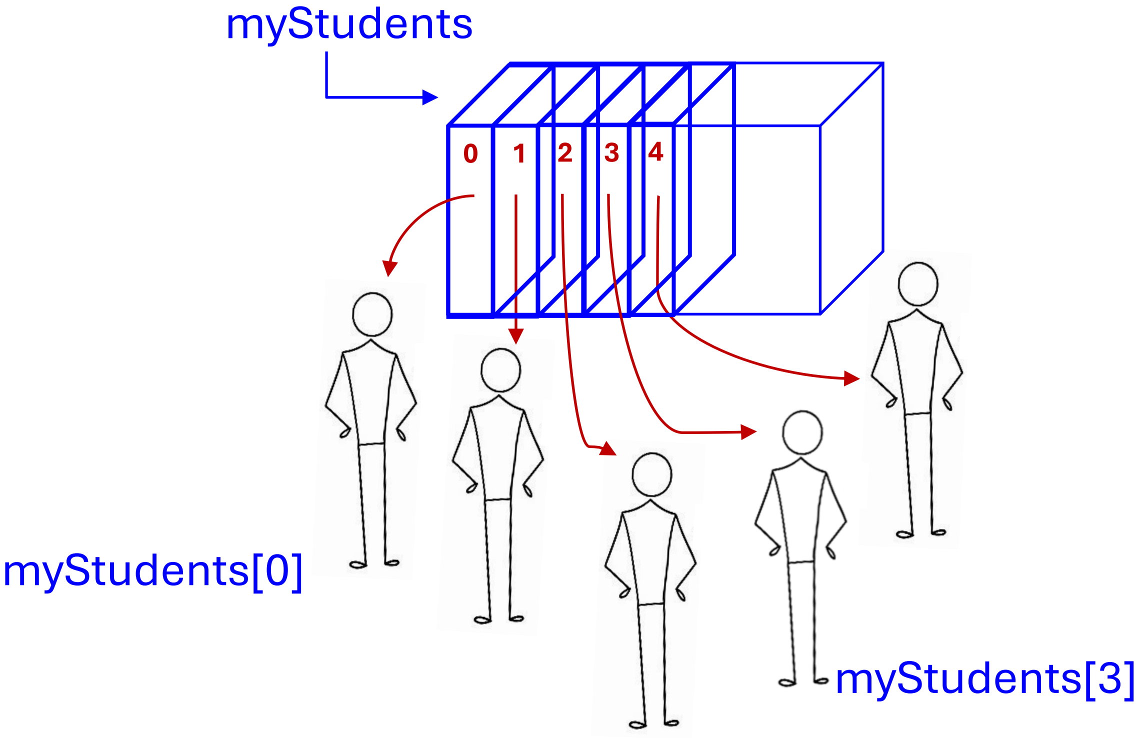


```
person1 = Person("Joe", "Blog", 2.56)
person2 = Person("Mary", "Smith", 1.40)
person3 = Person("Sam", "Black", 1.65)
person4 = Person("Jack", "Moore", 1.67)
```

It gets tedious to deal with them as individual variables.

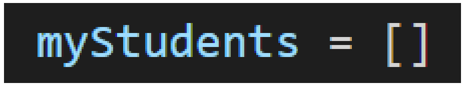
Lists

- List can be used to keep track of many objects.
- Think of a list like a card catalogue.
- Each card serves as a reference to an object.
- The cards are numbered starting at 0.

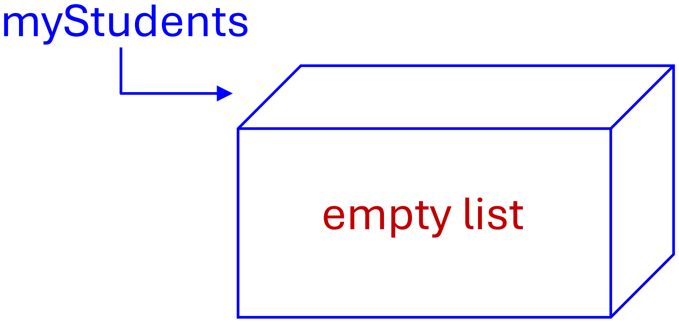


Managing Objects with Lists

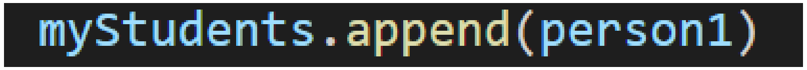
- Declaring a List Object



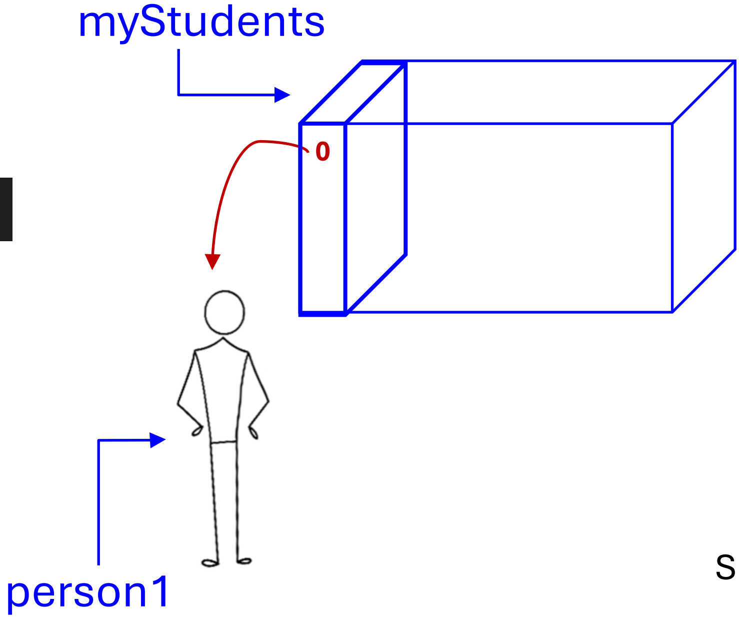
```
myStudents = []
```



- Adding Objects to the List



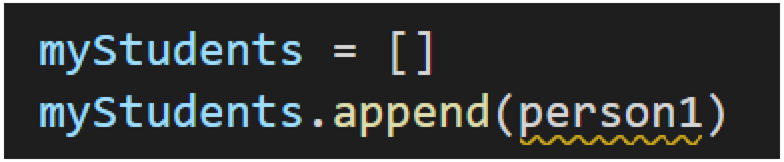
```
myStudents.append(person1)
```



Managing Objects with Lists

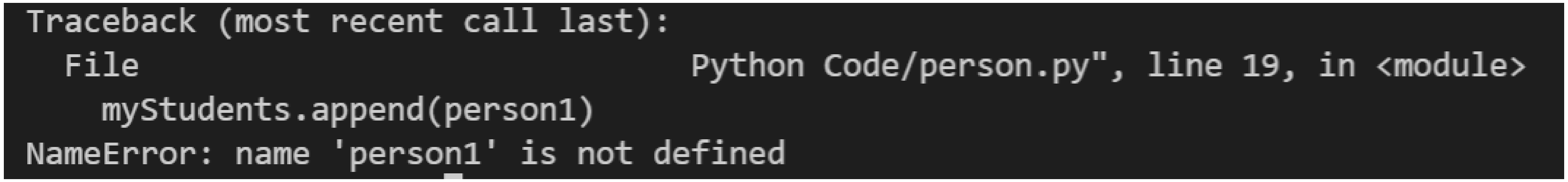
- Adding Objects to the List

– Don’t forget to instantiate the objects before trying to add it to the list.



```
myStudents = []
myStudents.append(person1)
```

You will get an error if you try to add an object that has not been created yet



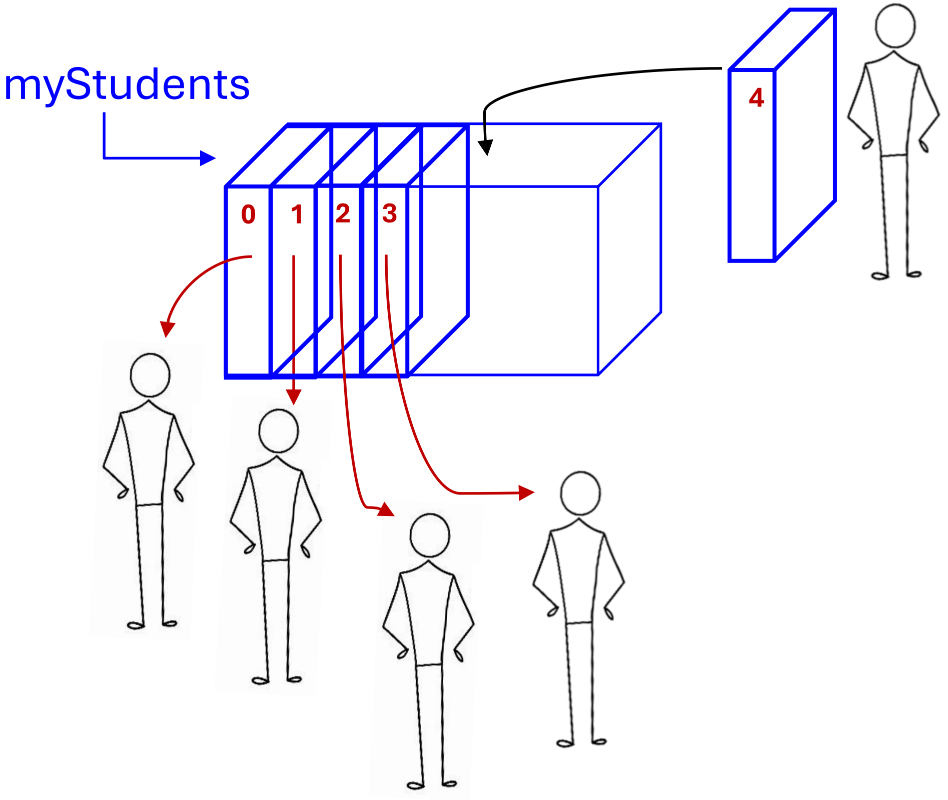
```
Traceback (most recent call last):
  File                                Python Code/person.py", line 19, in <module>
    myStudents.append(person1)
NameError: name 'person1' is not defined
```

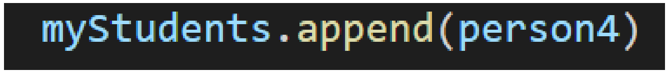
Managing Objects with Lists

- Adding Objects to the List

– The default is for objects to get added to the end of the list.

– For example, if students are at index 0, 1, 2, and 3, the next added student will be at index 4.





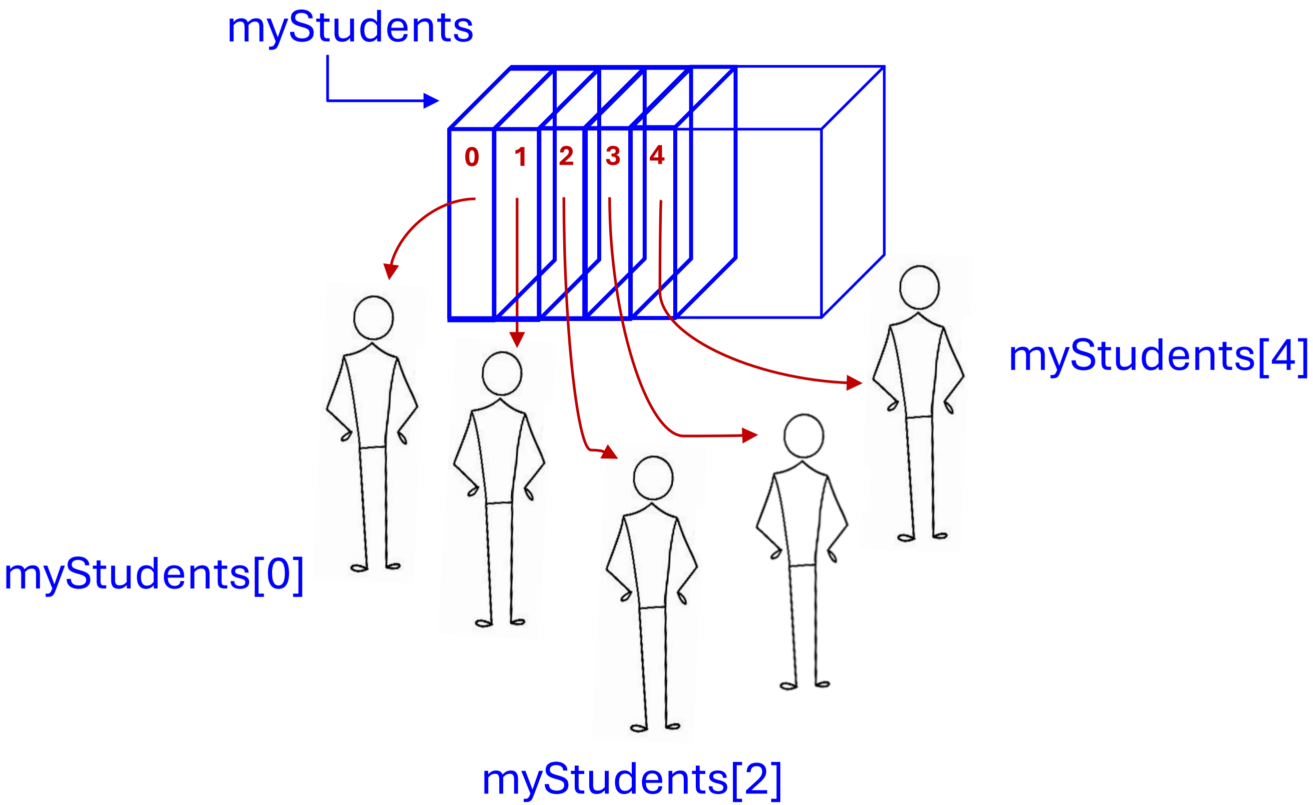
```
myStudents.append(person4)
```

append adds reference to the next object at the end of the list.

Managing Objects with Lists

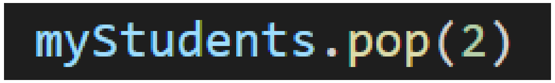
- Referring to Objects in the List

– Objects can be accessed by their index number in the list.

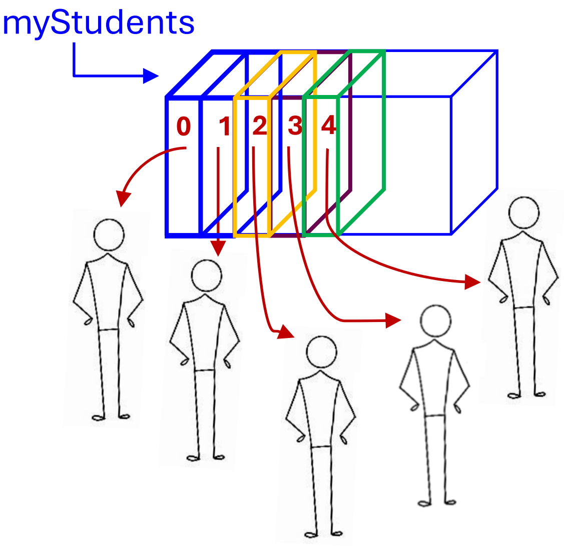


Managing Objects with Lists

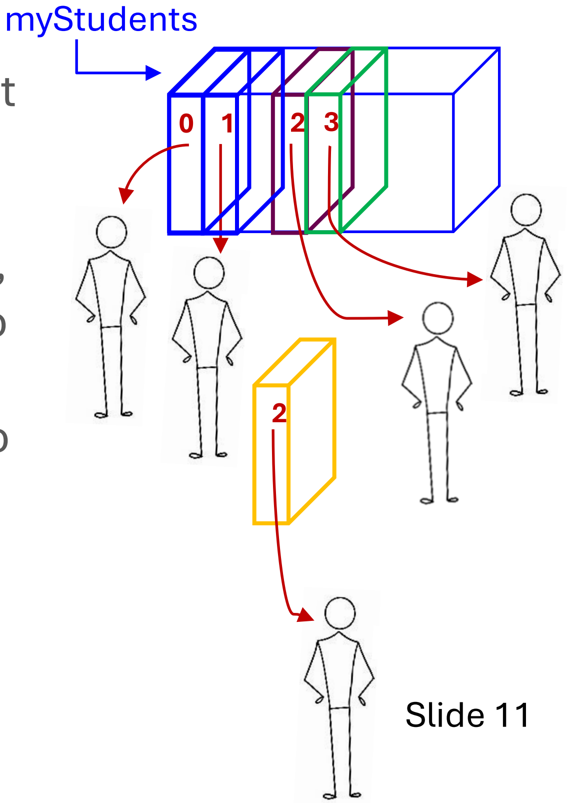
- Removing Objects from the List
  - The **pop** method removes an object at a specific index.
  - The objects following the removed one shift one index position forward.



```
myStudents.pop(2)
```



This removes the student at index 2.
All subsequent students (originally at indices 3, 4, etc.) shift one position to the left, so index 3 becomes index 2, and so on.

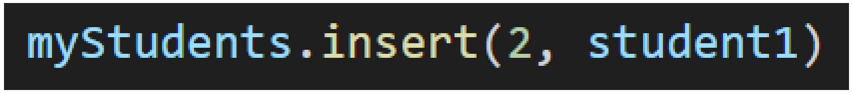


Managing Objects with Lists

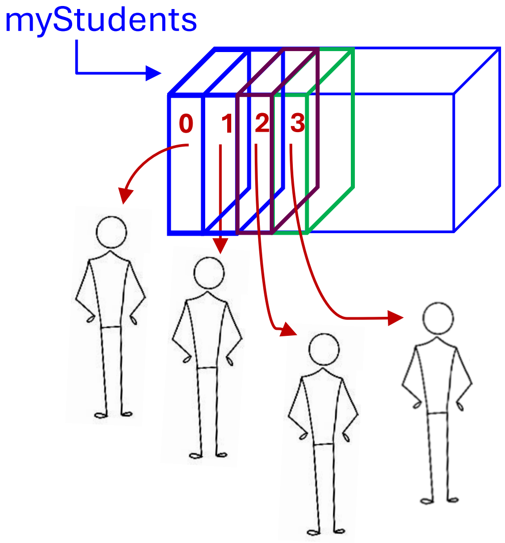
- Adding Objects to the List

– The **insert** method adds an object at a specified index.

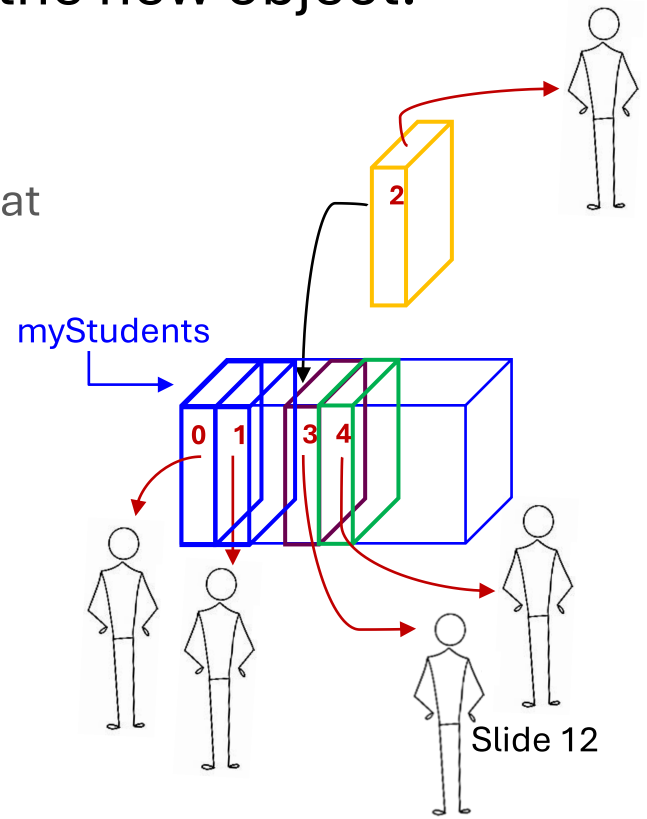
– All objects currently at and beyond that index shift one position to the right, making room for the new object.



```
myStudents.insert(2, student1)
```

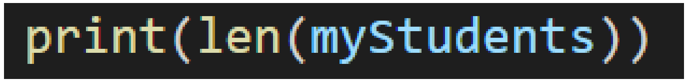


This inserts the student at index 2.
All students originally at indices 2 and beyond shift one position to the right. Index 2 now holds student1, and the students that were at indices 2, 3, etc., are moved to indices 3, 4, and so on.

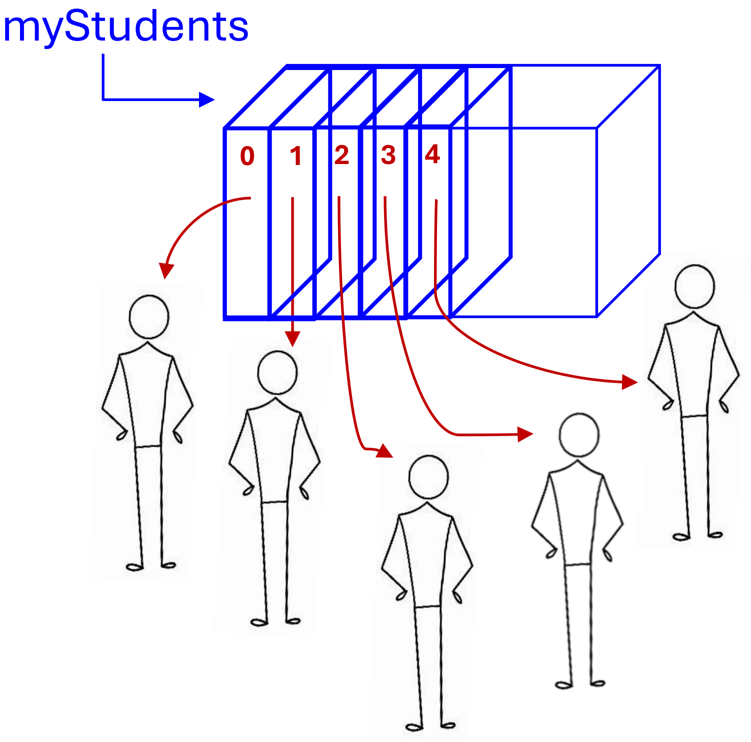


Other List Methods

- len() method returns the number of objects in the list.

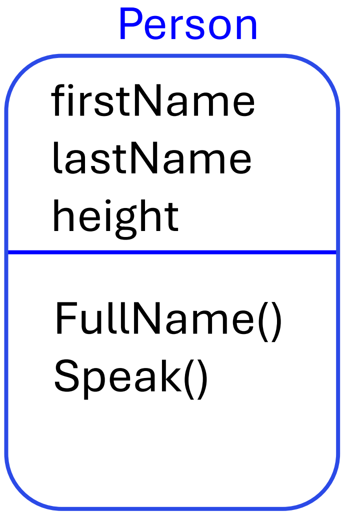


```
print(len(myStudents))
```



Exercise

Complete the tasks in **person _exercise1.py** to manage objects using a list.



Person

firstName
lastName
height

FullName()
Speak()

Test Your Knowledge

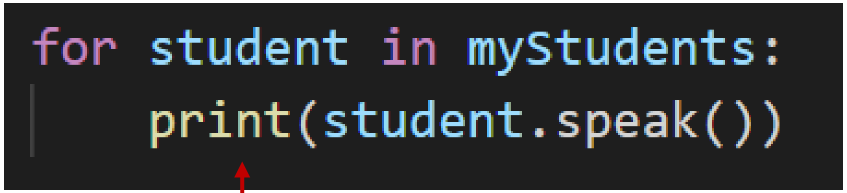
Write a method to add a new person to myStudents list.

Managing Objects with Lists

- Accessing All Objects in the List

– Use a construct called **for** loop to access each object in the list and invoke their methods to perform operations.

– For example, to write out greetings of all the students in the list, use a for loop to iterate through each object and call their speak method.

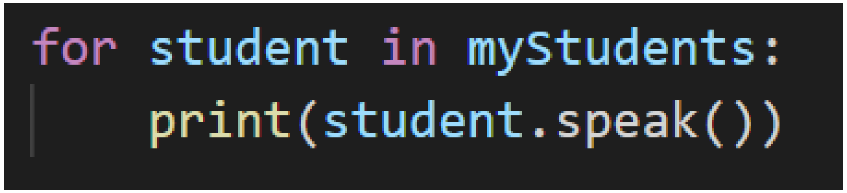


```python
for student in myStudents:
    print(student.speak())
```

This will be done for each person in the myStudents list.

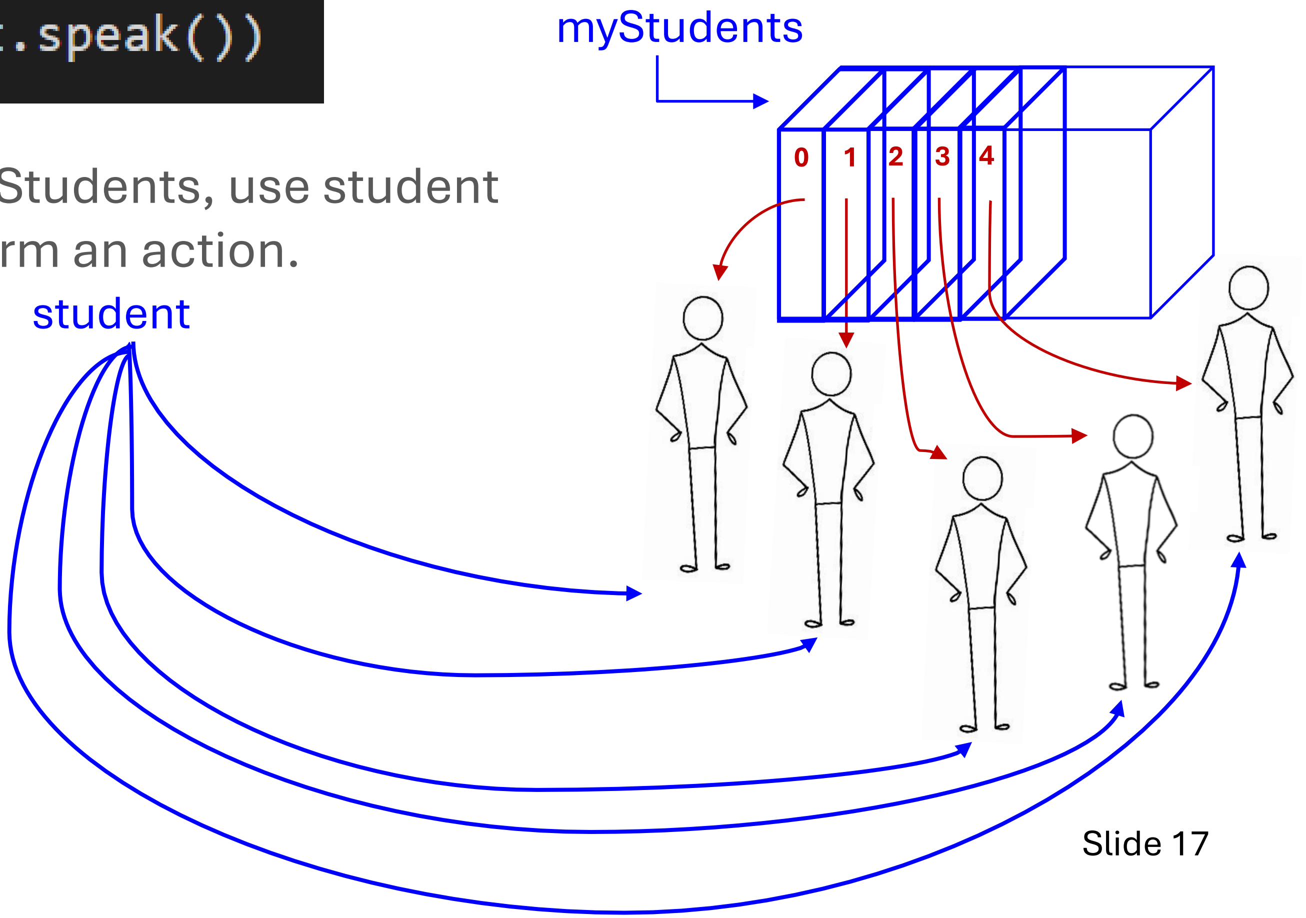
Managing Objects with Lists

- Accessing All Objects in the List

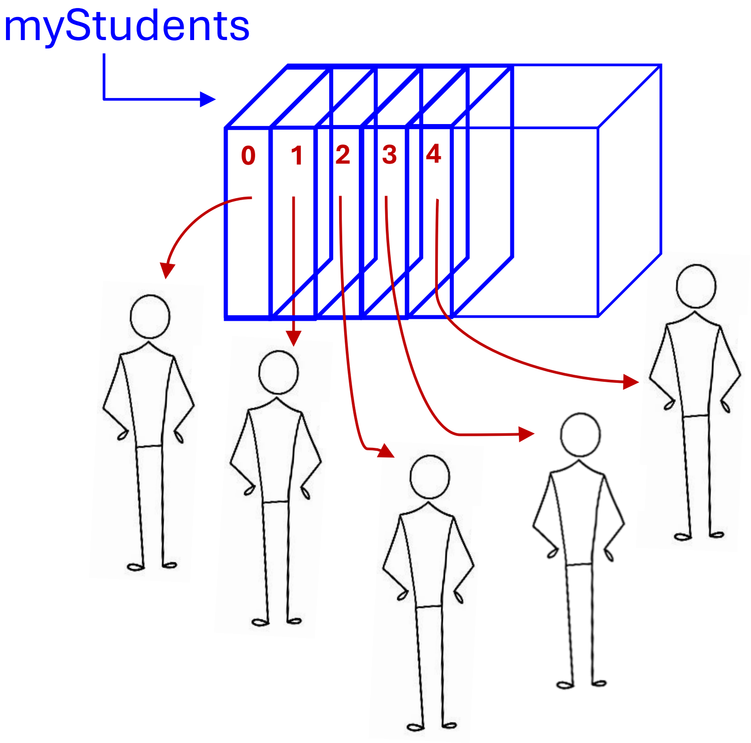


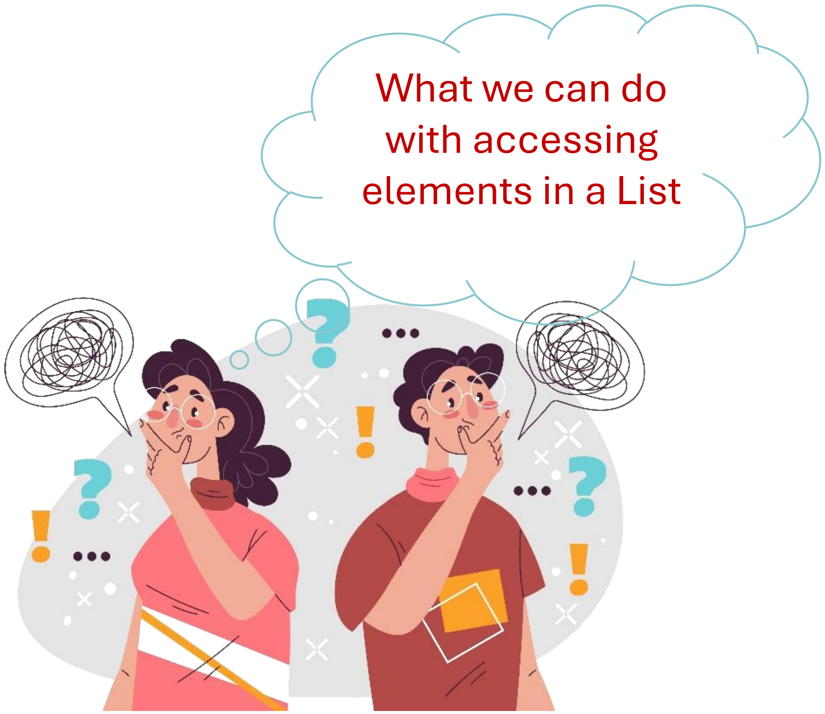
```
for student in myStudents:
    print(student.speak())
```

For each object in myStudents, use student to refer to it and perform an action.



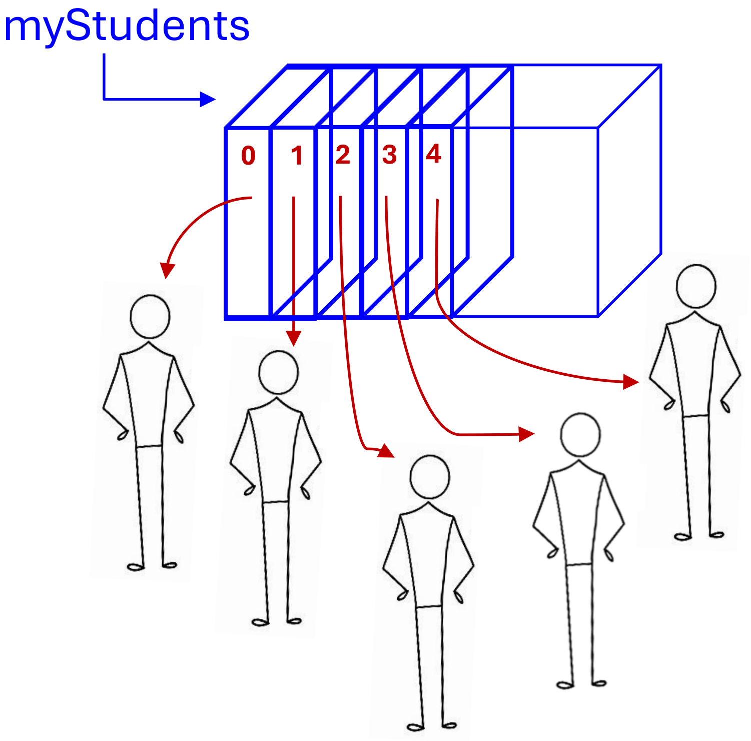
Working with List Elements

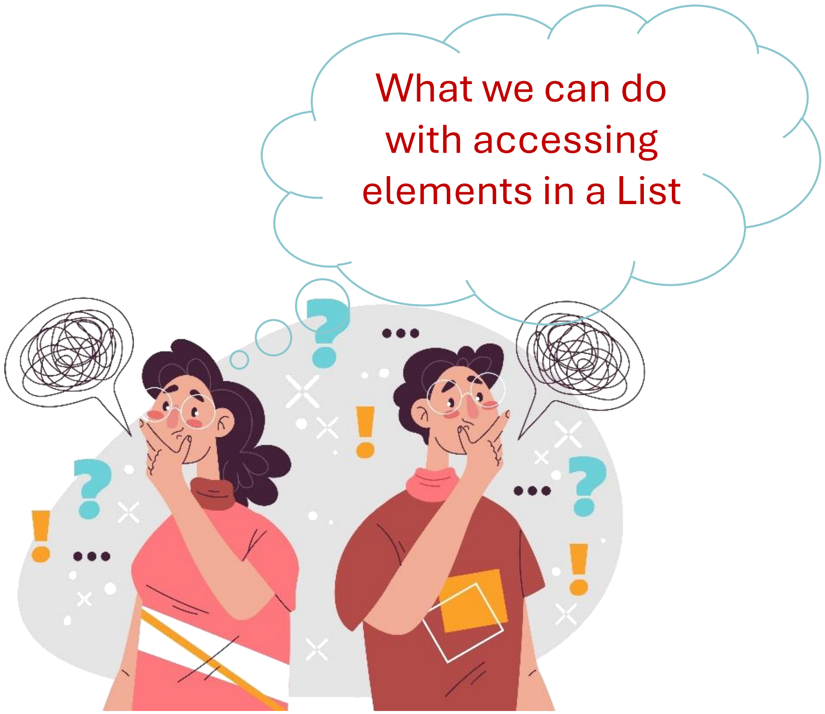




What we can do with accessing elements in a List

Working with List Elements

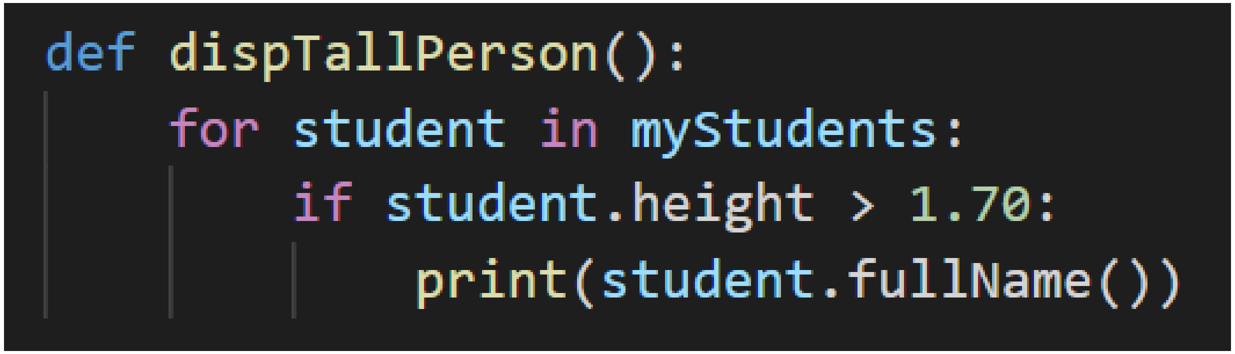



What we can do with accessing elements in a List

- Write out greetings for all persons
- Write out all names
- Write out all tall persons (> 1.70 meters)
- Count all tall persons

Working with List Elements

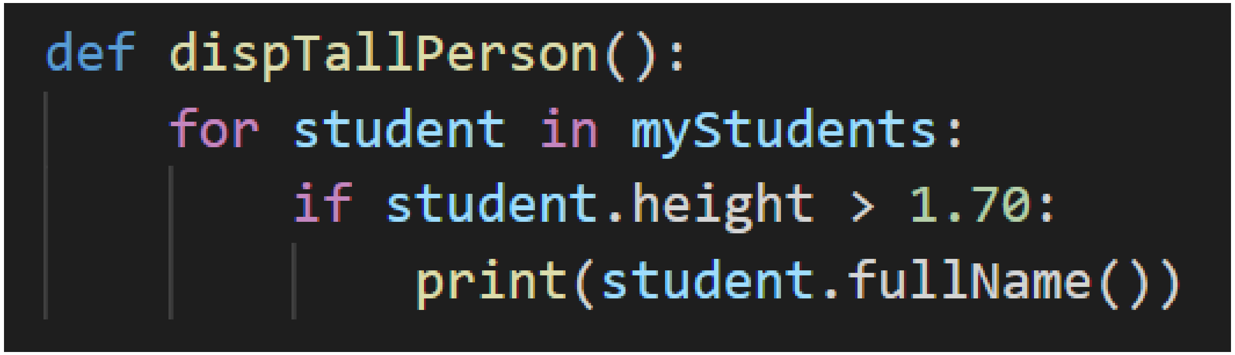
Write out all tall persons (> 1.70 meters)



```
def dispTallPerson():
    for student in myStudents:
        if student.height > 1.70:
            print(student.fullName())
```

Working with List Elements

How can we make the method more general to write out persons taller than any given height?

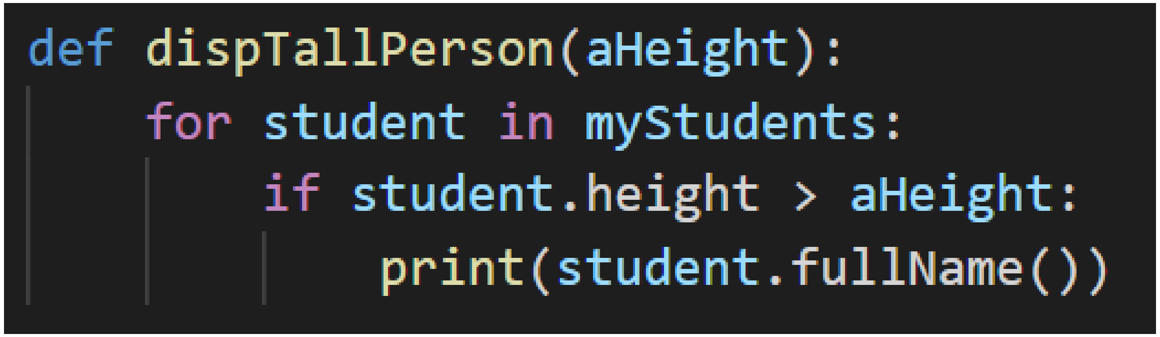


```
def dispTallPerson():
    for student in myStudents:
        if student.height > 1.70:
            print(student.fullName())
```

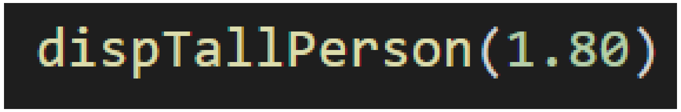
Working with List Elements

How can we make the method more general to write out persons taller than any given height?

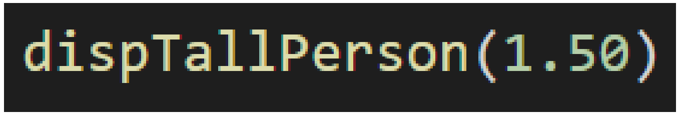
- Add a height parameter in the method definition.
- Modify the method to use this parameter to compare against each person's height.



```python
def dispTallPerson(aHeight):
    for student in myStudents:
        if student.height > aHeight:
            print(student.fullName())
```



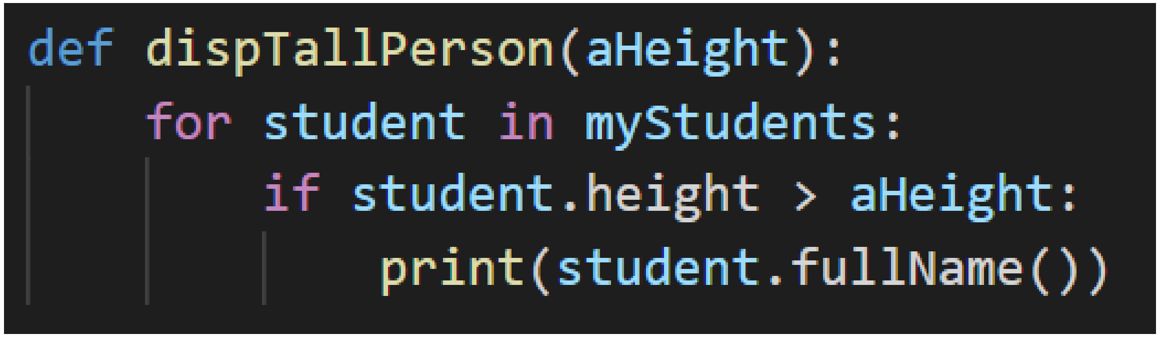
```python
dispTallPerson(1.80)
```



```python
dispTallPerson(1.50)
```

Working with List Elements

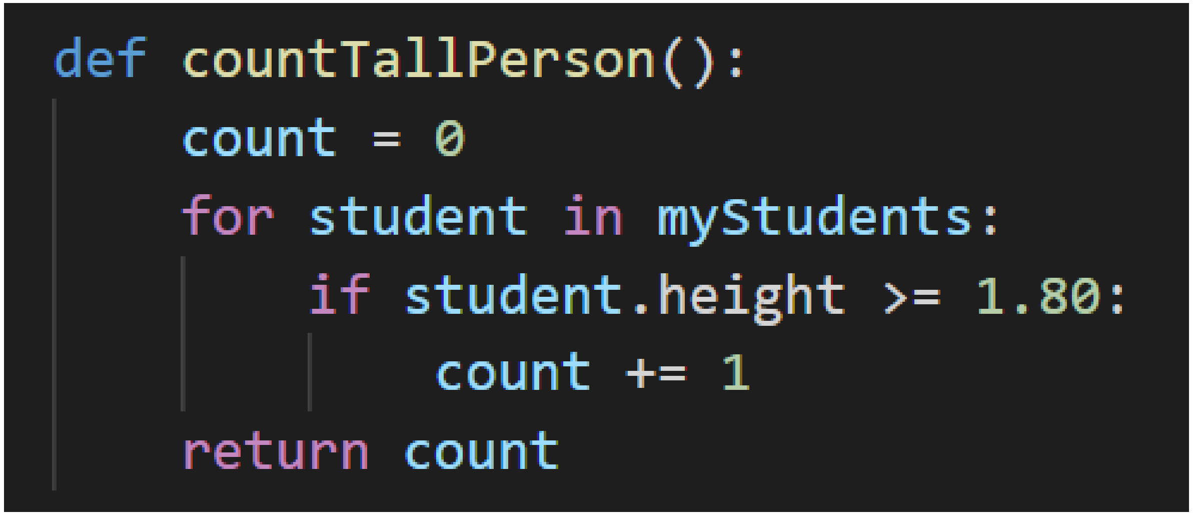
How can we modify the method to write out persons taller than any given height from a specified list of persons?



```python
def dispTallPerson(aHeight):
    for student in myStudents:
        if student.height > aHeight:
            print(student.fullName())
```

Working with List Elements

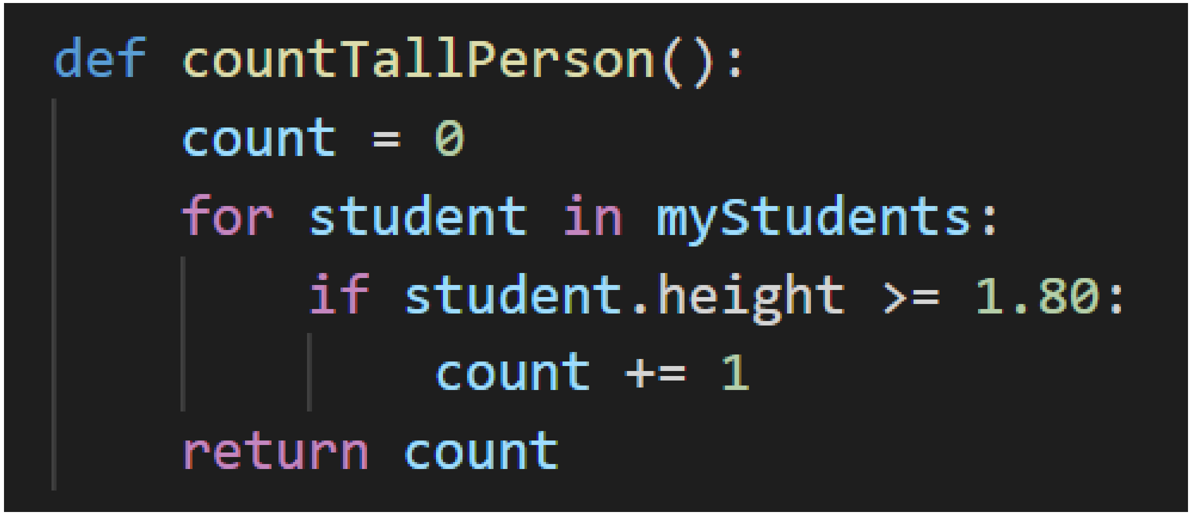
Count how many tall persons



```
def countTallPerson():
    count = 0
    for student in myStudents:
        if student.height >= 1.80:
            count += 1
    return count
```

Working with List Elements

How can we modify the method to count how many persons are taller than any given height from a specified list of persons?



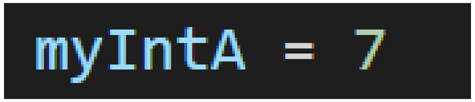
```python
def countTallPerson():
    count = 0
    for student in myStudents:
        if student.height >= 1.80:
            count += 1
    return count
```

Value Types and Reference Types

Value Types

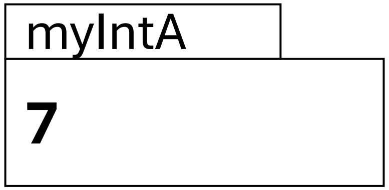
- Value types are simple types like integers, doubles, etc.
- The value is stored directly in the address of the variable.
- Example:

Creating a variable and assigning it a value



```
myIntA = 7
```

Storage in memory

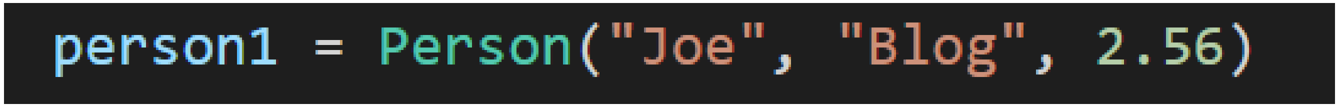


Since an integer value 7 is assigned to myIntA, the variable myIntA now holds the value 7.

Reference Types

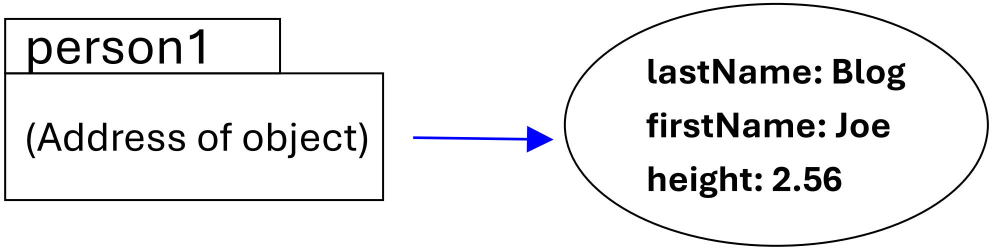
- Reference types hold references to the memory address where the actual data is stored.
- The variable stores a reference to the location in memory where the data resides, not the data itself.
- Example:

Creating a variable and assigning it an object created from a class



```
person1 = Person("Joe", "Blog", 2.56)
```

Storage in memory



person1

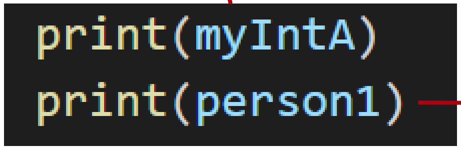
(Address of object)

lastName: Blog
firstName: Joe
height: 2.56

Since a Person object is assigned to person1, the variable person1 now holds a reference to the memory address where the object is stored.

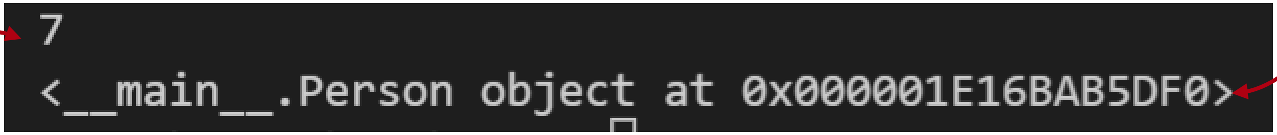
Value vs. Reference Types

myIntA contains actual value



```
print(myIntA)
print(person1)
```

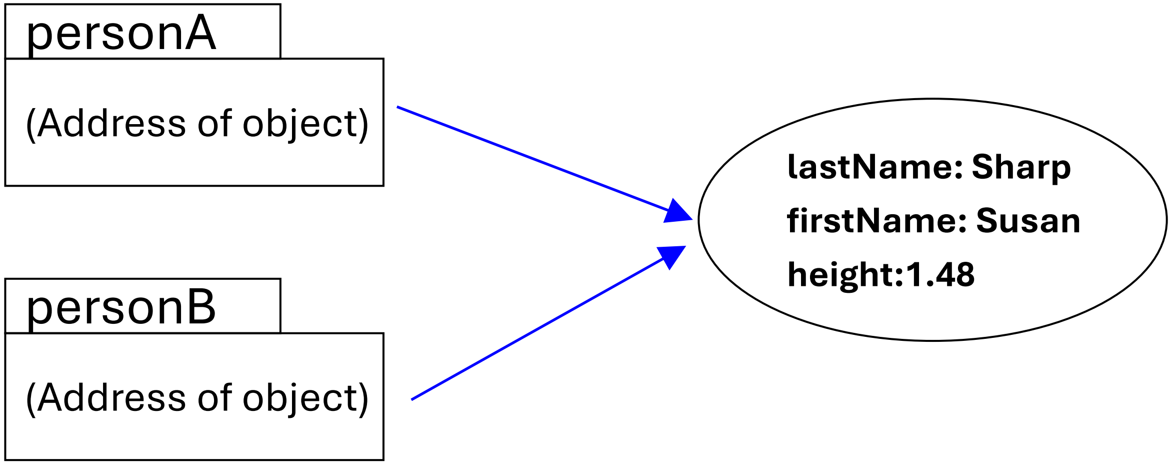
person1 is an address (reference)



```
7
<__main__.Person object at 0x000001E16BAB5DF0>
```

Multiple References to a Single Object

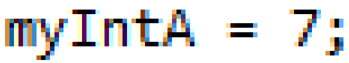
- An object in memory can be referenced by more than one variable.
- Each variable holds a reference to the same memory location where the object is stored.
- Example:



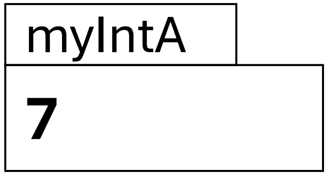
This is like a person being called by different nicknames or aliases.

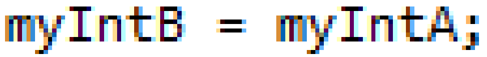
Assignment Statements for Value Types

- For value types, an assignment statement copies the actual value into the new variable.
- Example:

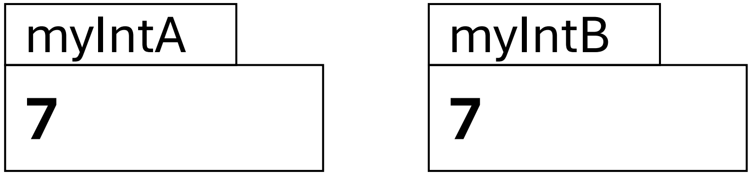


```
myIntA = 7;
```





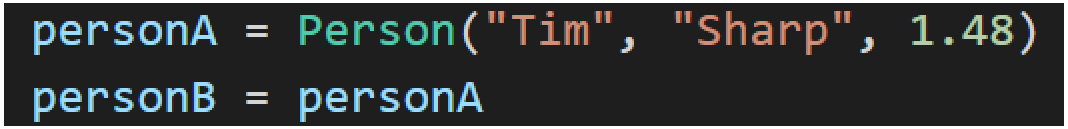
```
myIntB = myIntA;
```



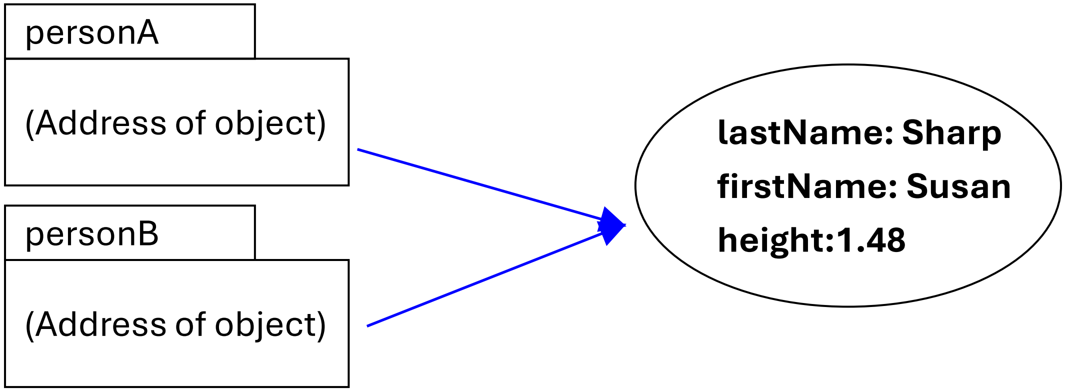
The value (e.g., 7) is stored independently in each variable, allowing them to be changed independently.

Assignment Statements for Reference Types

- For reference types (objects), an assignment statement means making the variables **refer** to the same object.
- Example:



```
personA = Person("Tim", "Sharp", 1.48)
personB = personA
```



personA

(Address of object)

personB

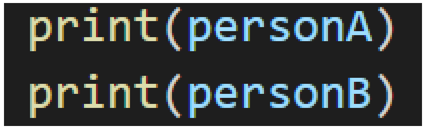
(Address of object)

lastName: Sharp

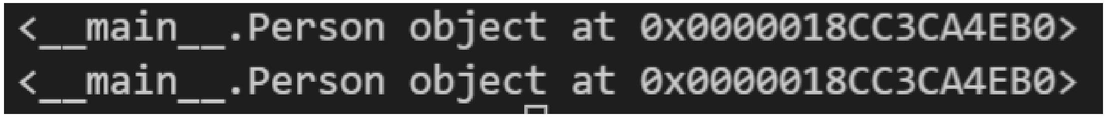
firstName: Susan

height:1.48

Both personA and personB refer to the same address.



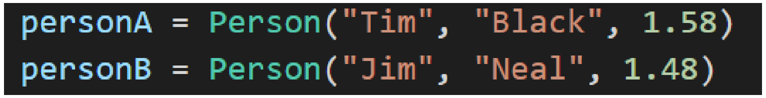
```
print(personA)
print(personB)
```



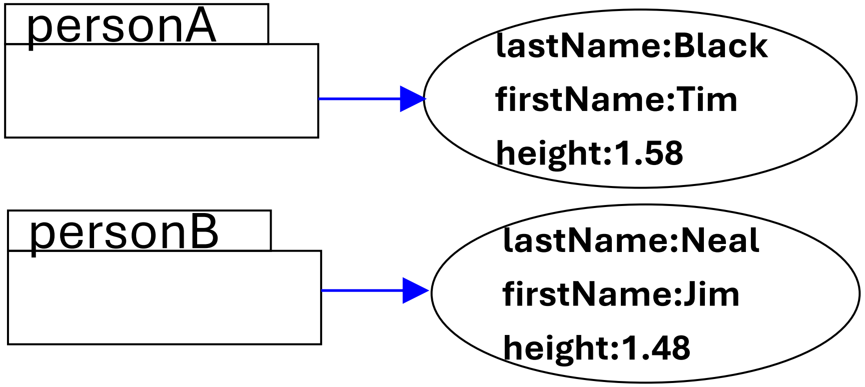
```
<__main__.Person object at 0x0000018CC3CA4EB0>
<__main__.Person object at 0x0000018CC3CA4EB0>
```

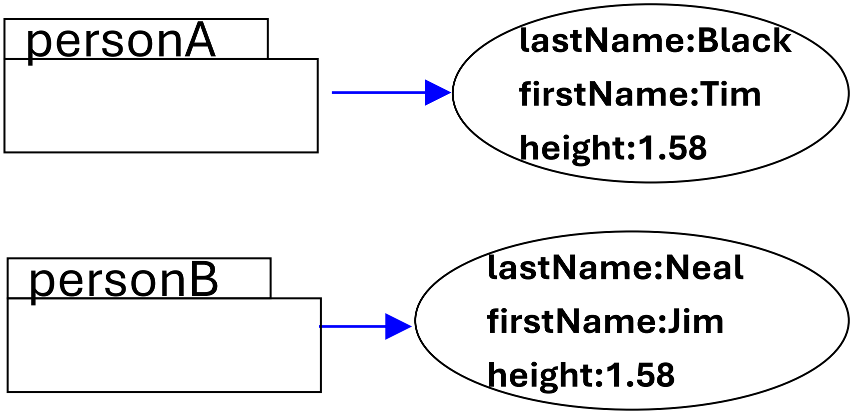
Assignment Statements for Reference Types

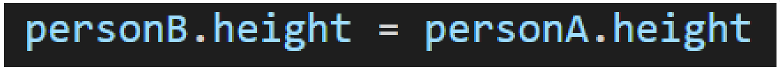
- Example:



```
personA = Person("Tim", "Black", 1.58)
personB = Person("Jim", "Neal", 1.48)
```





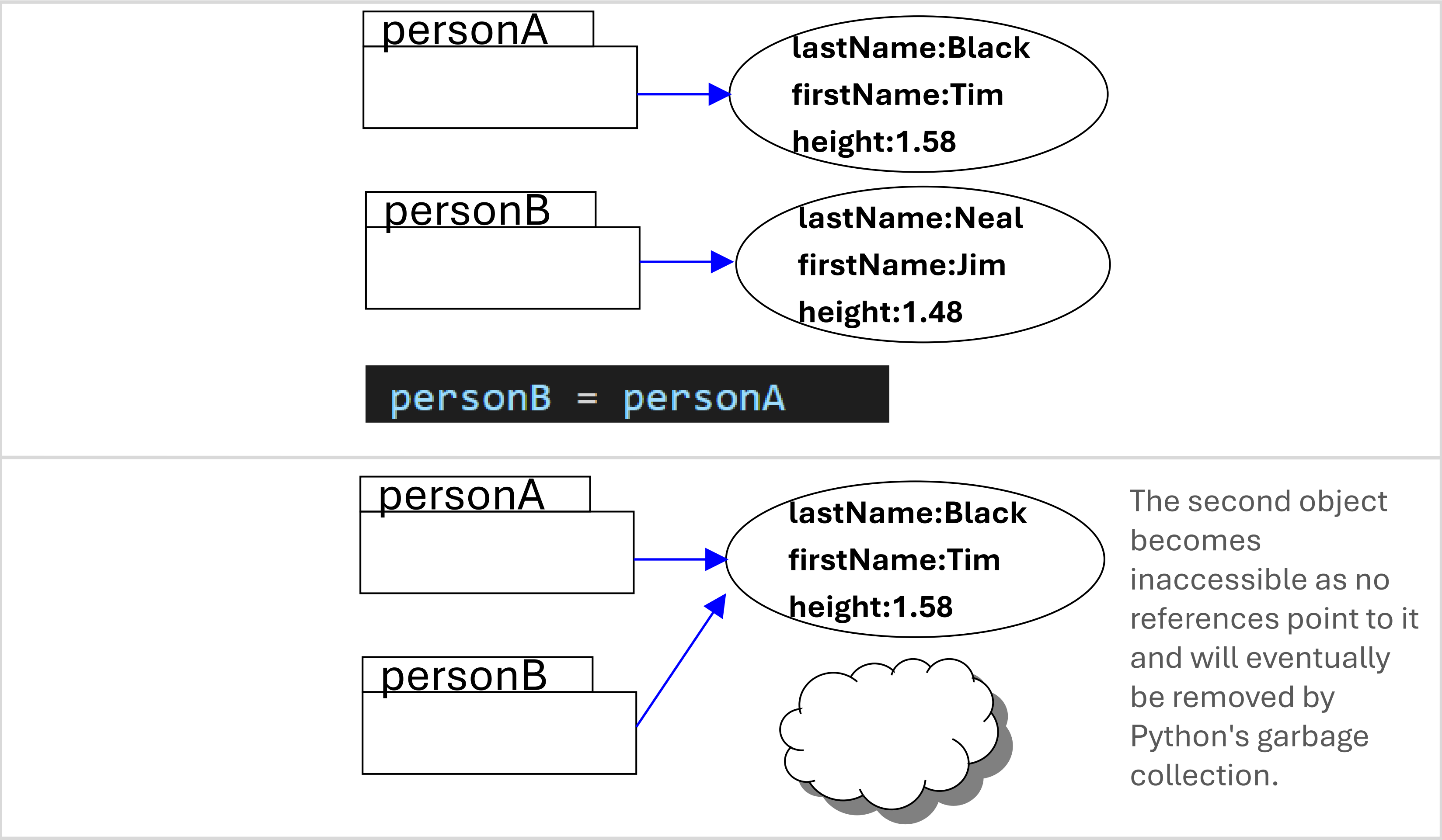


```
personB.height = personA.height
```

Set personB's height to be the same as personA's height.

Assignment Statements for Reference Types

- Example:

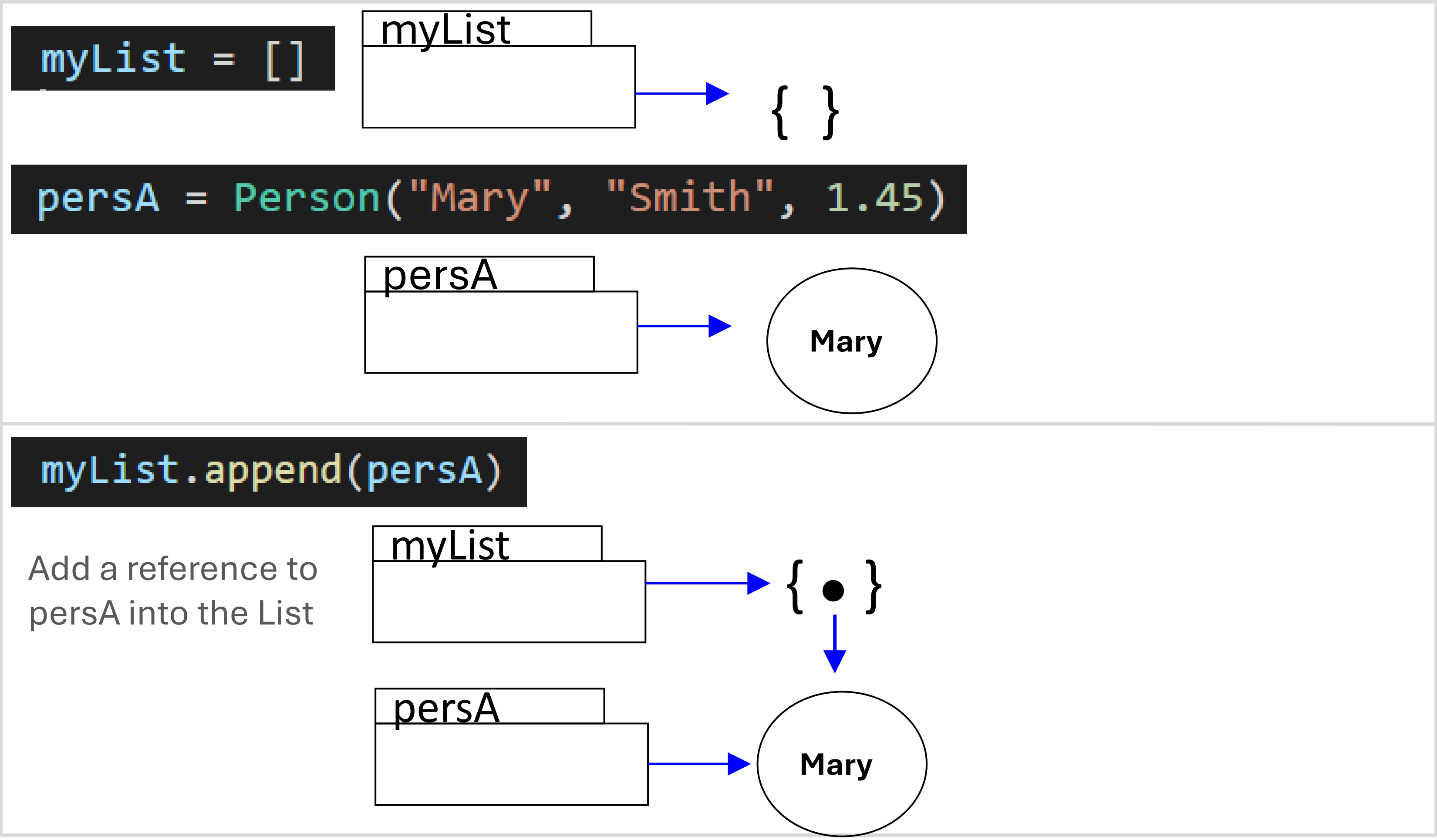


```
personB = personA
```

The second object becomes inaccessible as no references point to it and will eventually be removed by Python's garbage collection.

Adding References to a List

- A list is simply a collection of references pointing to objects.



```
myList = []
```

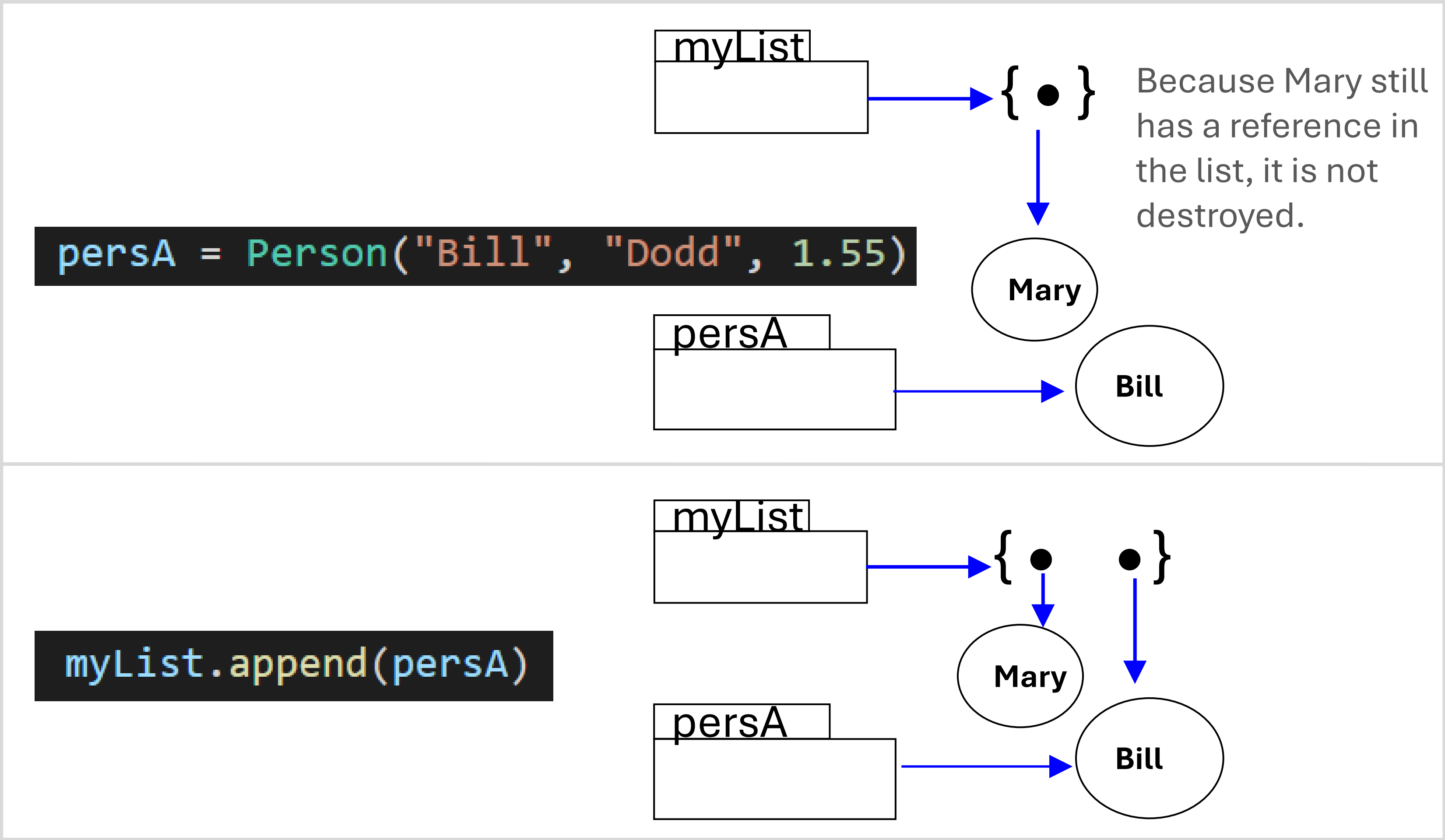
```
persA = Person("Mary", "Smith", 1.45)
```

```
myList.append(persA)
```

Add a reference to persA into the List

Adding References to a List

Adding the next object



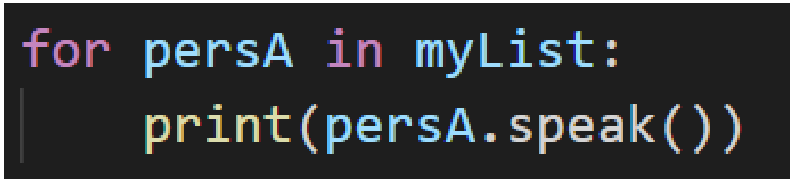
```
persA = Person("Bill", "Dodd", 1.55)
```

```
myList.append(persA)
```

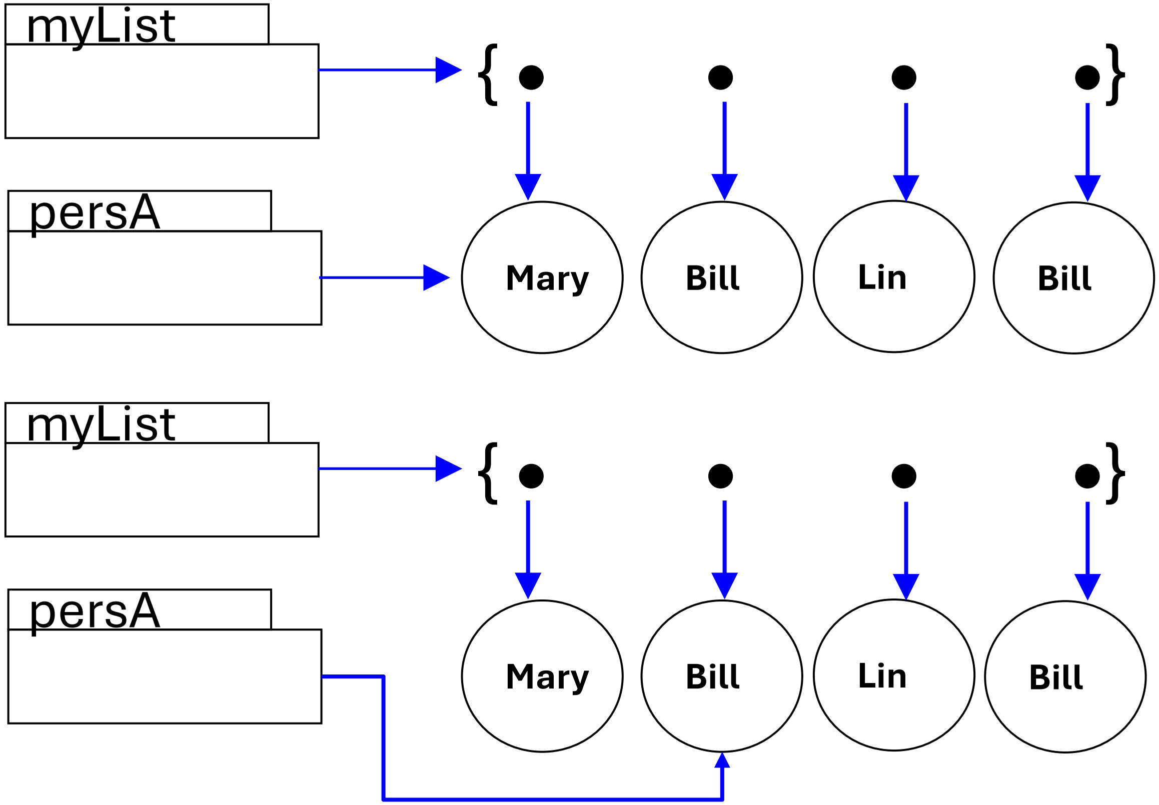
Because Mary still has a reference in the list, it is not destroyed.

Iterating Over a List

- The variable in the **for** loop refers to each object in turn.



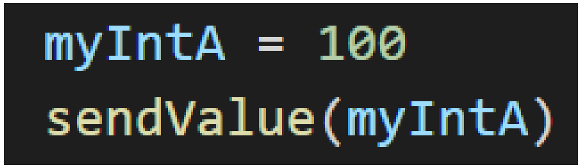
```
for persA in myList:
    print(persA.speak())
```



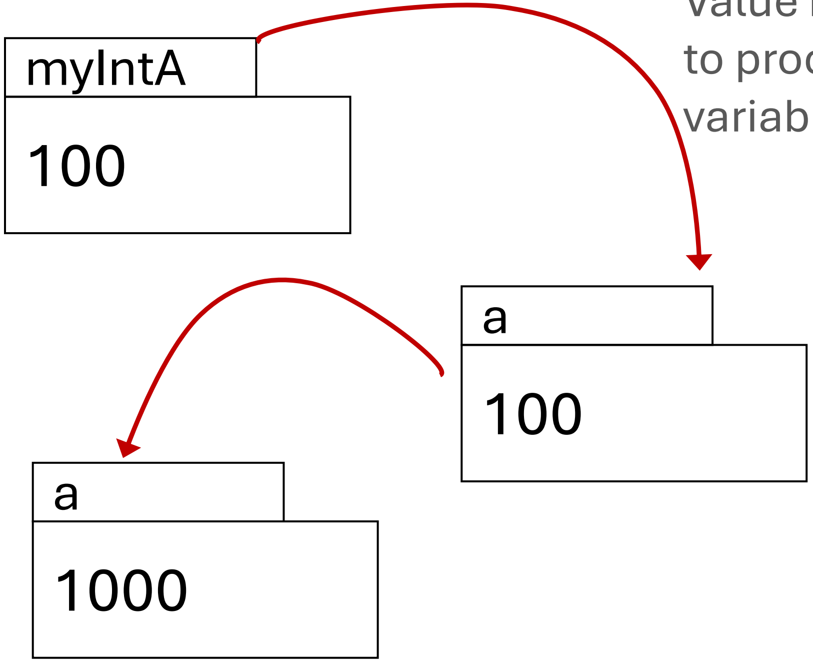
Using Value Types and Reference Types as Parameters

Values as Parameters

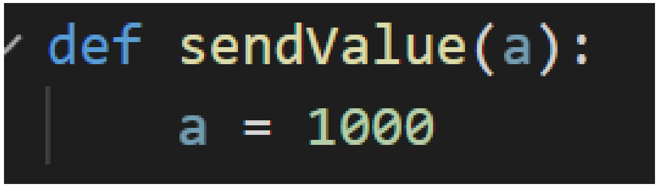
- The value of the parameter is copied into the procedure’s own variable.
- Any changes affect only the copy, not the original variable.
- The original value of the variable remains unchanged.
- Example:



```
myIntA = 100
sendValue(myIntA)
```



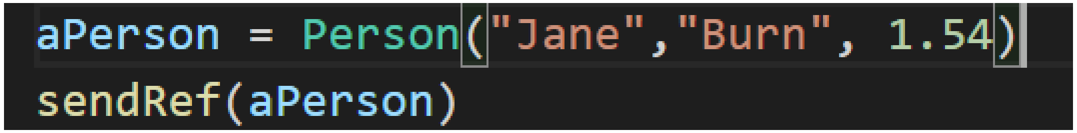
Value is **copied** to procedure’s variable



```
def sendValue(a):
    a = 1000
```

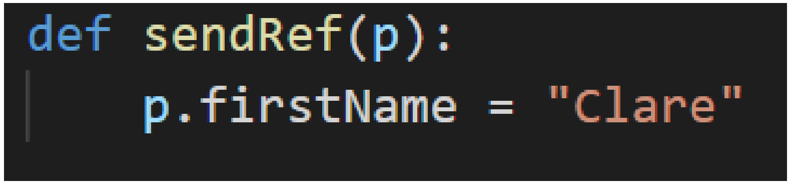
References as Parameters

- The parameter holds a reference to the original object in memory.
- Changes made to the parameter affect the original object.
- Both the original variable and the parameter refer to the same object in memory.
- Example:

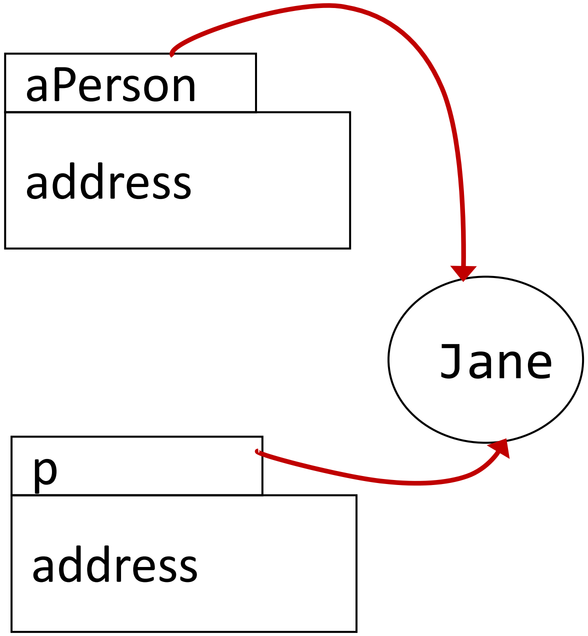


```
aPerson = Person("Jane","Burn", 1.54)
sendRef(aPerson)
```

Address is copied to procedure’s variable

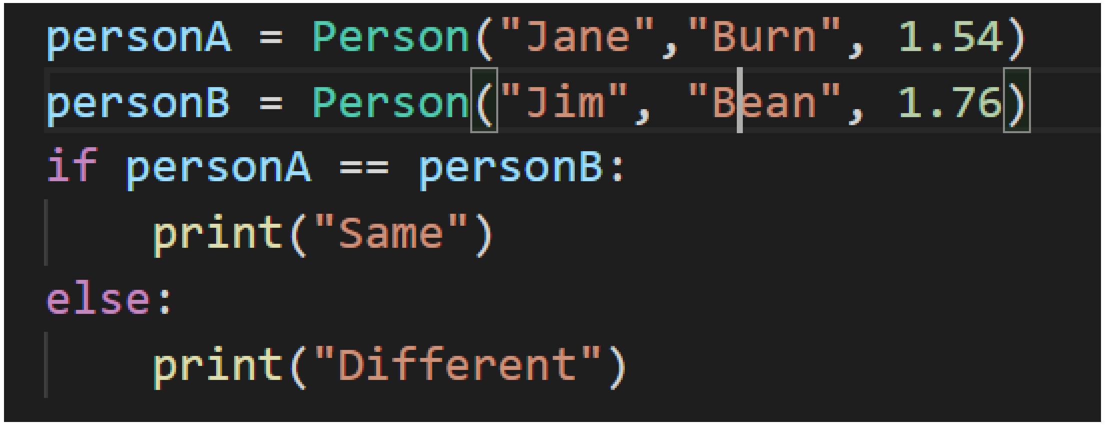


```
def sendRef(p):
    p.firstName = "Clare"
```



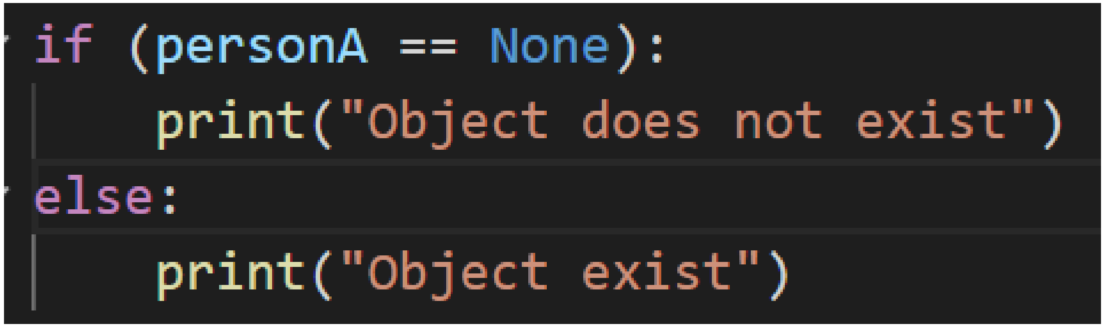
Comparing Object References

- Use == to check if two references point to the same object.
- Example:



```
personA = Person("Jane","Burn", 1.54)
personB = Person("Jim", "Bean", 1.76)
if personA == personB:
    print("Same")
else:
    print("Different")
```

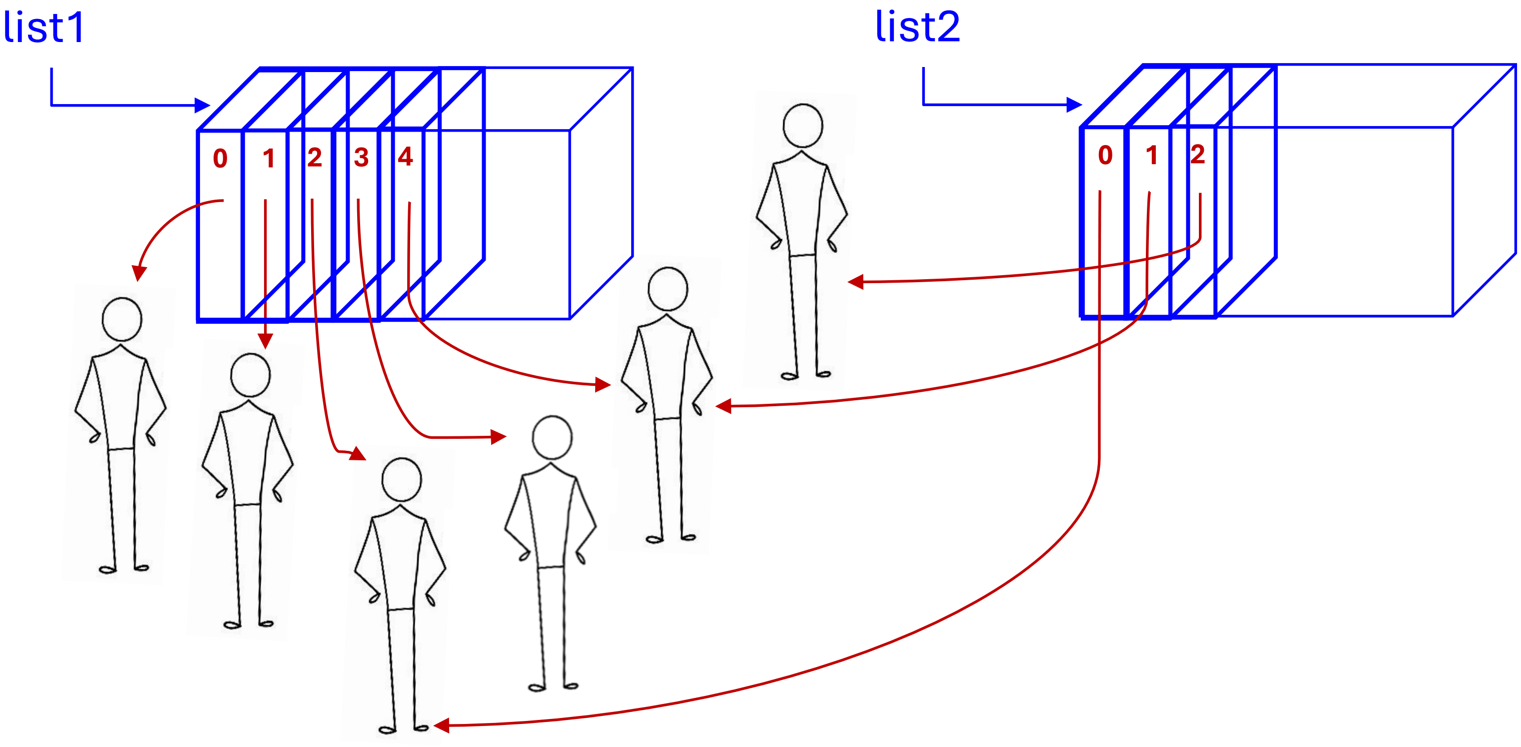
Check whether a variable refers to anything at all



```
if (personA == None):
    print("Object does not exist")
else:
    print("Object exist")
```

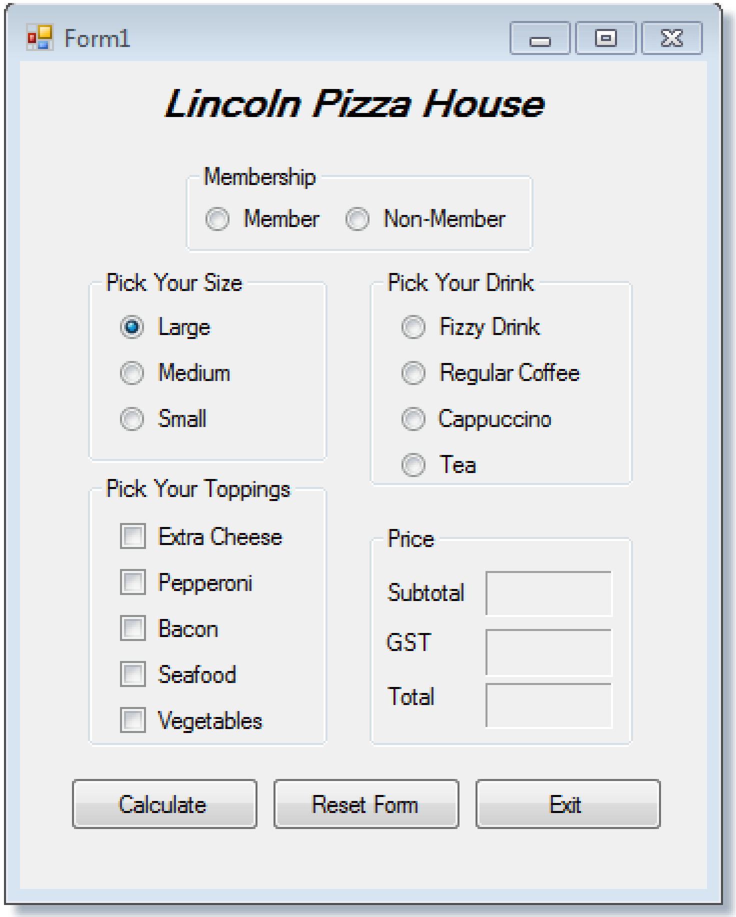
Exercise 2

Complete the task in **person _exercise2.py** by writing out the names of all Person objects that are present in both list1 and list2.



Test Your Knowledge

Develop a system for Lincoln Pizza House using OOP principles. Consider how you would handle processing multiple pizza orders in an object-oriented approach.



Lincoln Pizza House

Membership

○ Member ○ Non-Member

Pick Your Size

● Large

○ Medium

○ Small

Pick Your Drink

○ Fizzy Drink

○ Regular Coffee

○ Cappuccino

○ Tea

Pick Your Toppings

□ Extra Cheese

□ Pepperoni

□ Bacon

□ Seafood

□ Vegetables

Price

Subtotal

GST

Total