



Models in Memory in Python



Roberto A. Bittencourt



Retrieve

Retrieve: Naïve Algorithm

- ▶ Choose the search field.
- ▶ Enter the object search string.
- ▶ Search the string in the objects' search field and create either a collection of object indices or of objects themselves that satisfy the search criterion.
- ▶ Return the collection.

Retrieve: Improved Algorithm

- ▶ Enter the object search string.
- ▶ Search the string in the objects' search field and create either a collection of object indices or of objects themselves that satisfy the search criterion.
- ▶ If any object was found:
 - ▶ Return the collection
- ▶ If no object was found:
 - ▶ Return an empty collection

Retrieve: Program in Python with an *unordered list* or with an *ordered list*

```
def retrieve(self, name):
    retrieved = []
    for element in self.students:
        if name in element.name:
            retrieved.append(element)
    return retrieved

def main():
    school = School('Central High')
    school.create('Peter Smith', 123, 1995)
    school.create('Rika Nakamura', 345, 1987)
    school.create('Peter Parker', 234, 1998)
    retrieved = school.retrieve('Peter')
    if retrieved:
        print("Retrieved students:")
        for student in retrieved:
            print(student)
    else:
        print("Students with name %d not found.\n\n" % name)
```

Retrieve: Program in Python with a *dictionary*

```
def retrieve(self, name):
    retrieved = []
    for value in self.students.values():
        if name in value.name:
            retrieved.append(element)
    return retrieved

def main():
    school = School('Central High')
    school.create('Peter Smith', 123, 1995)
    school.create('Rika Nakamura', 345, 1987)
    school.create('Peter Parker', 234, 1998)
    retrieved = school.retrieve('Peter')
    if retrieved:
        print("Retrieved students:")
        for student in retrieved:
            print(student)
    else:
        print("Students with name %d not found.\n\n" % name)
```

Testing School: retrieve students (school_test.py)

```
def test_retrieve(self):
    student1 = Student('Peter Jackson', 123, 2010)
    student2 = Student('Peter Parker', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    school = School('Central High')
    self.assertEqual(0, len(school.retrieve('Peter')), "empty student collection")

    # add some students
    school.create('Peter Jackson', 123, 2010)
    school.create('Peter Parker', 234, 2011)
    school.create('Kala', 345, 2009)

    # Retrieve a singleton list
    retrieved = school.retrieve('Kala')
    self.assertEqual(1, len(school.retrieve('Kala')),
                     "there should be 1 student named Kala")
    self.assertEqual(student3, retrieved[0], "Retrieving Kala")

    # Retrieve a list with more than one element
    retrieved = school.retrieve('Peter')
    self.assertEqual(2, len(school.retrieve('Peter')),
                     "there should be 2 students named Peter")
    self.assertEqual(student1, retrieved[0], "Retrieving Peter Jackson")
    self.assertEqual(student2, retrieved[1], "Retrieving Peter Parker")
```



Update

Update: Naïve Algorithm

1. Enter the object key.
2. Search the key and return either the object index or the object itself.
3. Update the object.

Update: Improved Algorithm

1. Enter the object key.
2. Search the key and return either the object index or the object itself.
3. If the object was found:
 1. If the key will change with the update operation:
 1. Check whether the key needs to be updated in the collection
 2. Update the object
 3. Return success
4. If the object was not found:
 1. Return failure

Update: Program in Python with an *unordered list* or with an *ordered list*

```
def update(self, key, name, number, birth_year):
    student = self.search(key)
    if student:
        student.name = name
        student.number = number # simpler in lists
        student.birth_year = birth_year
    return True
else:
    return False

def main():
    school = School('Central High')
    school.create('Peter', 123, 1992)
    school.create('Rika', 345, 1987)
    school.create('Ali', 234, 1998)
    if school.update(123, 'Peter Jackson', 123, 1995):
        print('Peter updated')
    if school.update(234, 'Ali Mesbah', 102, 1998):
        print('Ali updated')
```

Update: Program in Python with a *dictionary*

```
def update(self, key, name, number, birth_year):
    student = self.search(key)
    if student:
        if key != number:
            student = self.students.pop(key)
            self.students[number] = student
        student.name = name
        student.number = number
        student.birth_year = birth_year
    return True
else:
    return False

def main():
    school = School('Central High')
    school.create('Peter', 123, 1992)
    school.create('Rika', 345, 1987)
    school.create('Ali', 234, 1998)
    if school.update(123, 'Peter Jackson', 123, 1995):
        print('Peter updated')
    if school.update(234, 'Ali Mesbah', 102, 1998):
        print('Ali updated')
```

Testing School: update student (school_test.py)

```
def test_update(self):
    student1 = Student('Peter', 123, 2010)
    student1a = Student('Peter Jackson', 123, 1995)
    student2 = Student('Ali', 234, 2011)
    student2a = Student('Ali Mesbah', 102, 2011)
    student3 = Student('Kala', 345, 2009)

    school = School('Central High')
    self.assertFalse(school.update(123, 'Peter Jackson', 123, 1995), "empty student collection")

    # add some students
    school.create('Peter', 123, 2010)
    school.create('Ali', 234, 2011)
    school.create('Kala', 345, 2009)

    # Do not update an unregistered student
    self.assertFalse(school.update(300, 'Latifa', 300, 2001), "Unregistered student")

    # Update a registered student, maintaining key
    self.assertTrue(school.update(123, 'Peter Jackson', 123, 1995), "Updating Peter, same key")

    # Update a registered student, changing key
    self.assertTrue(school.update(234, 'Ali Mesbah', 102, 1998), "Updating Ali, different key")
```



Delete

Delete: Naïve Algorithm

1. Enter the object key.
2. Search the key and return either the object index or the object itself.
3. Delete the object.

Delete: Improved Algorithm

1. Enter the object key.
2. Search the key and return either the object index or the object itself.
3. If the object was found:
 1. Delete the object
 2. Return success
4. If the object was not found:
 1. Return failure

Delete: Program in Python with an *unordered list* or with an *ordered list*

```
def delete(self, key):
    element = self.search(key)
    if element:
        self.students.remove(element)
        return True
    else:
        return False

def main():
    school = School('Central High')
    school.create('Peter', 123, 1992)
    school.create('Rika', 345, 1987)
    school.create('Ali', 234, 1998)
    if school.delete(123):
        print('Peter deleted')
    if school.delete(200):
        print('Student deleted')
    else:
        print('Error deleting student')
```

Delete: Program in Python with a *dictionary*

```
def delete(self, key):
    student = self.search(key)
    if student:
        self.students.pop(key)
        return True
    else:
        return False

def main():
    school.create('Peter', 123, 1992)
    school.create('Rika', 345, 1987)
    school.create('Ali', 234, 1998)
    if school.delete(123):
        print('Peter deleted')
    if school.delete(200):
        print('Student deleted')
    else:
        print('Error deleting student')
```

Testing School: delete student (school_test.py)

```
def test_delete(self):
    student1 = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    school = School('Central High')
    self.assertFalse(school.delete(123), "empty student collection")

    # add some students
    school.create('Peter', 123, 2010)
    school.create('Ali', 234, 2011)
    school.create('Kala', 345, 2009)

    # Do not delete an unregistered student
    self.assertFalse(school.delete(300), "Unregistered student")

    # Delete a registered student
    self.assertTrue(school.delete(123), "Deleting Peter")

    # Do not delete a student that has already been deleted
    self.assertFalse(school.delete(123), "Deleting Peter again should not be possible")

    # Delete another registered student
    self.assertTrue(school.delete(234), "Deleting Ali")
```