# MACS 30111

# Classes and Objects (examples)

# Misc

# Topics:

- Stock Span Example
- Class Inheritance in Python
- Python string formatting
- M/D/1 Queue Example

# Stock Span example

- **The stock span problem** is a financial problem where we have a series of **N** daily price quotes for a stock and we need to calculate the span of the stock's price for all N days. The span $S_i$ of the stock's price on a given day i is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than its price on the given day.

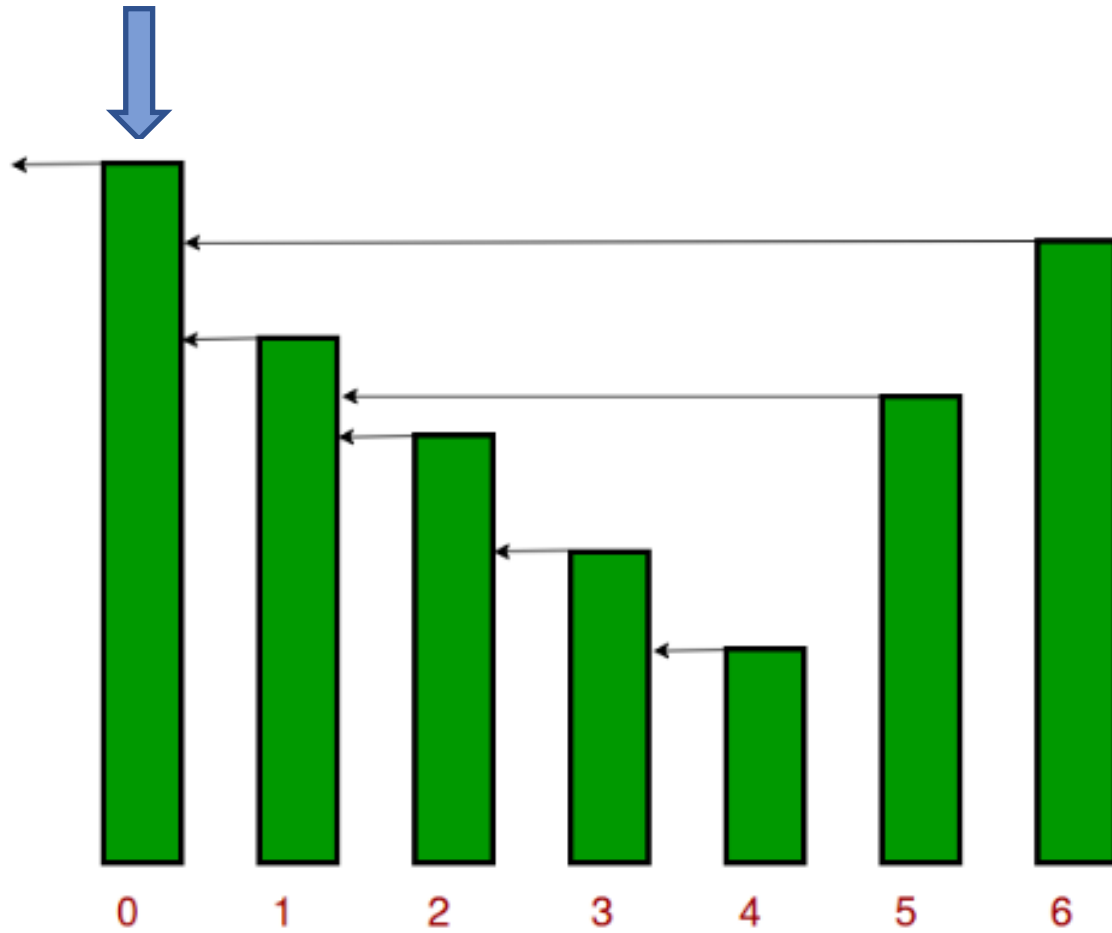- How could we represent this using our current skillset?

# Stock Span example

- Financial problem
- A series of n daily price quotes for a stock
- The span of the stock's price for all days:
  - $S\_i$: max consecutive days
  - Stock price less than or equal to the current day
  - Including the current day

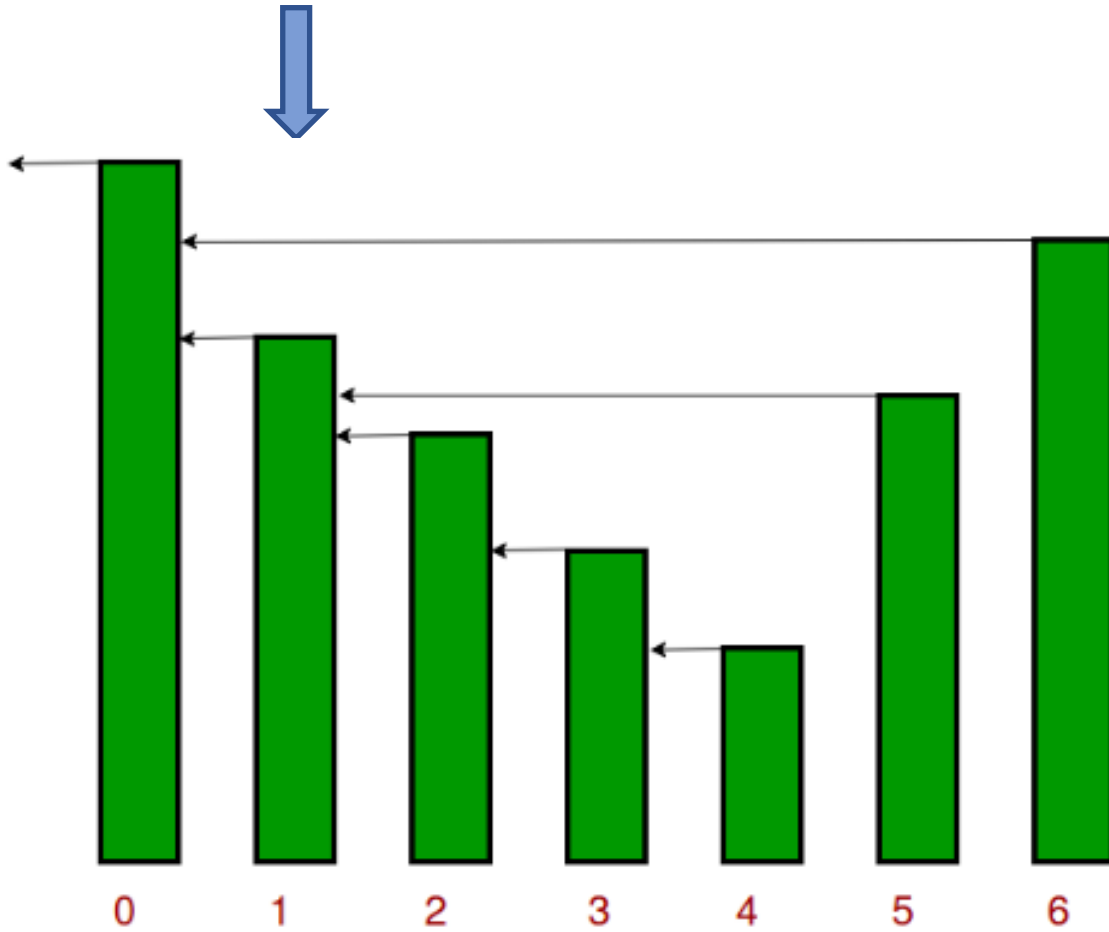- Price = [100, 80, 60, 70, 60, 75, 85]

List / stack implementation
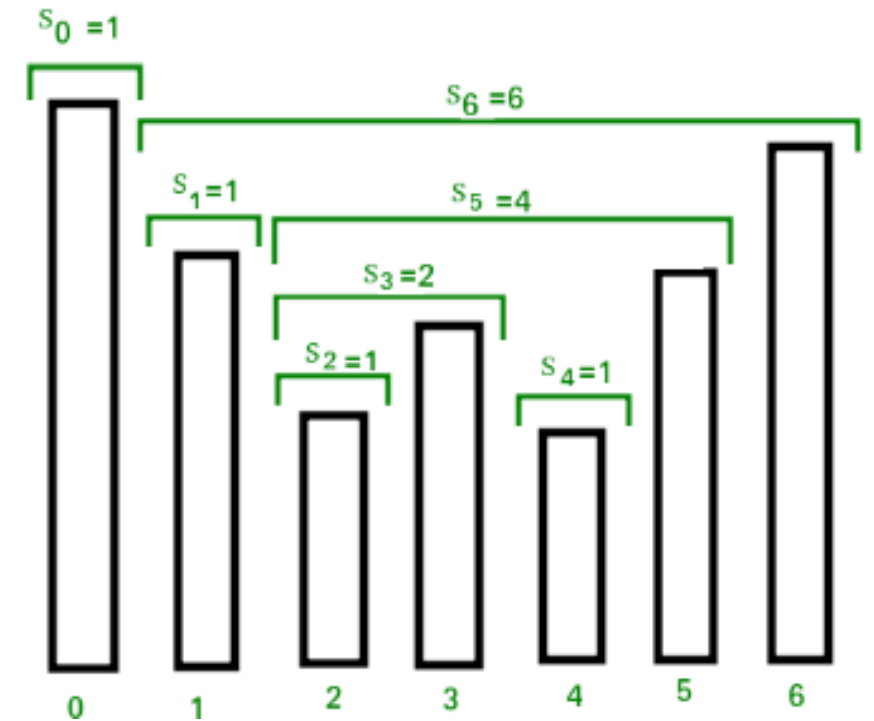


Reference: course website M4

```python
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```

```
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```

TOP

| 0 |

_____

BOTTOM

```python
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```
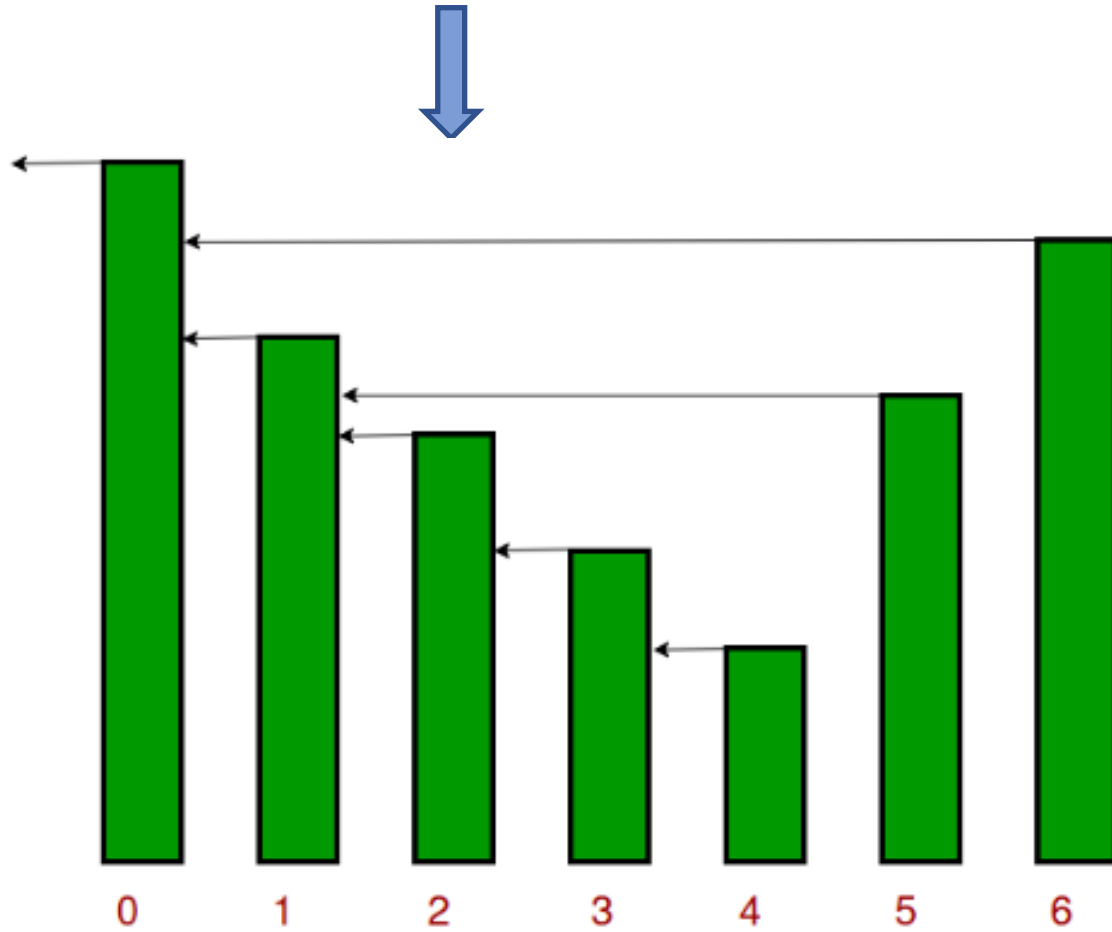
```python
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```

```
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```
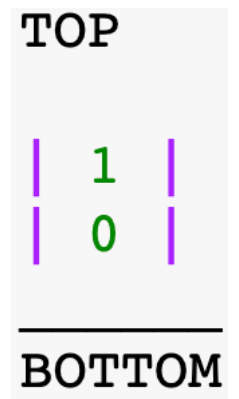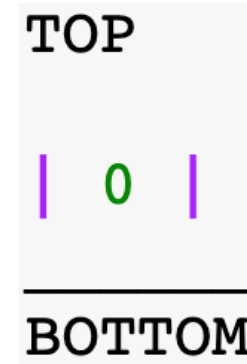
```python
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```

TOP

| 4 |
| 3 |
| 2 |
| 1 |
| 0 |

BOTTOM

TOP

| 1 |
| 0 |

BOTTOM

```python
for i in range(1, n):
    while( len(st) > 0 and price[st[-1]] <= price[i]):
        st.pop()

    if len(st) <= 0:
        S[i] = i + 1
    else:
        S[i] = i - st[-1]

    st.append(i)
```
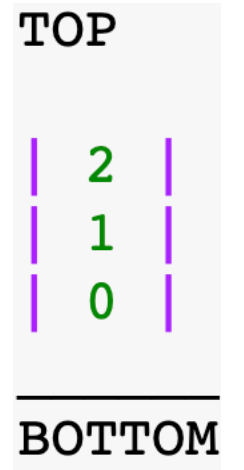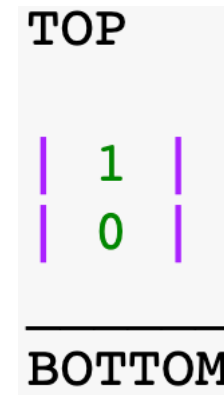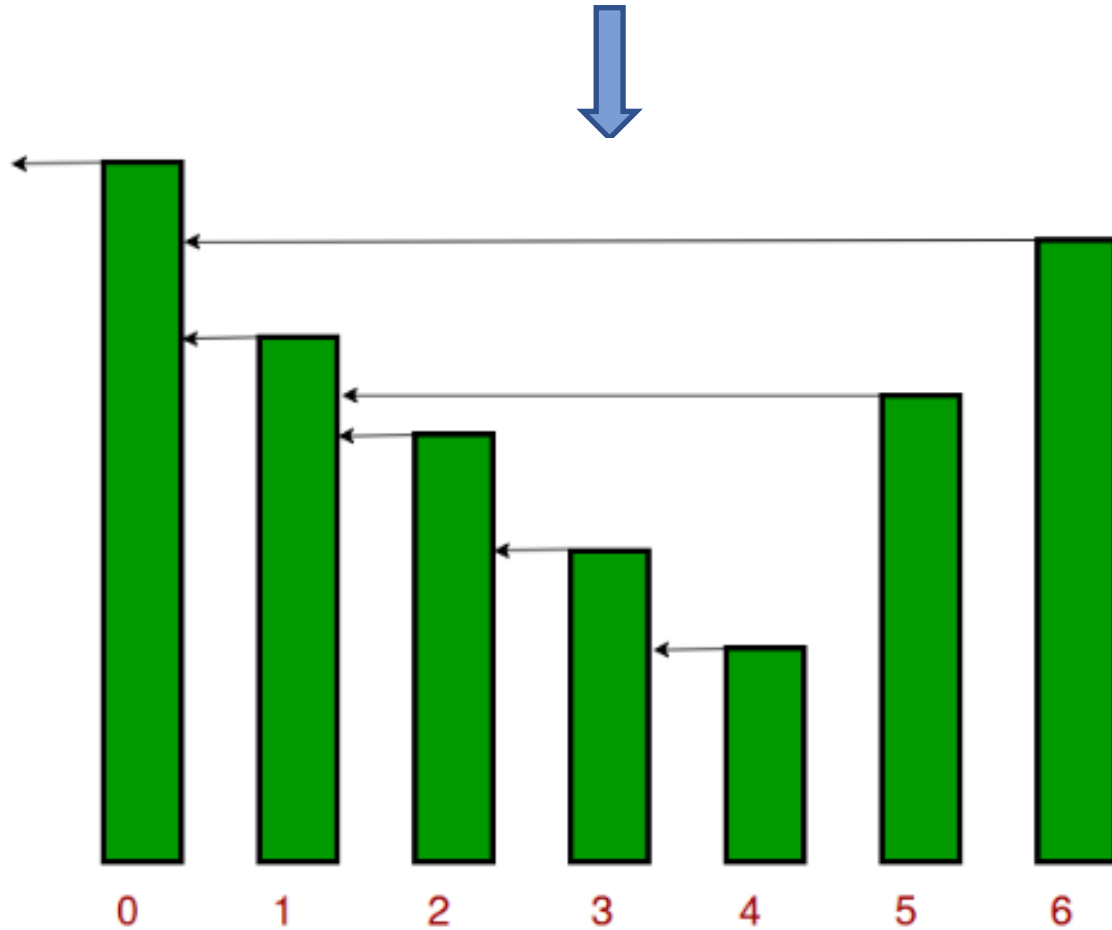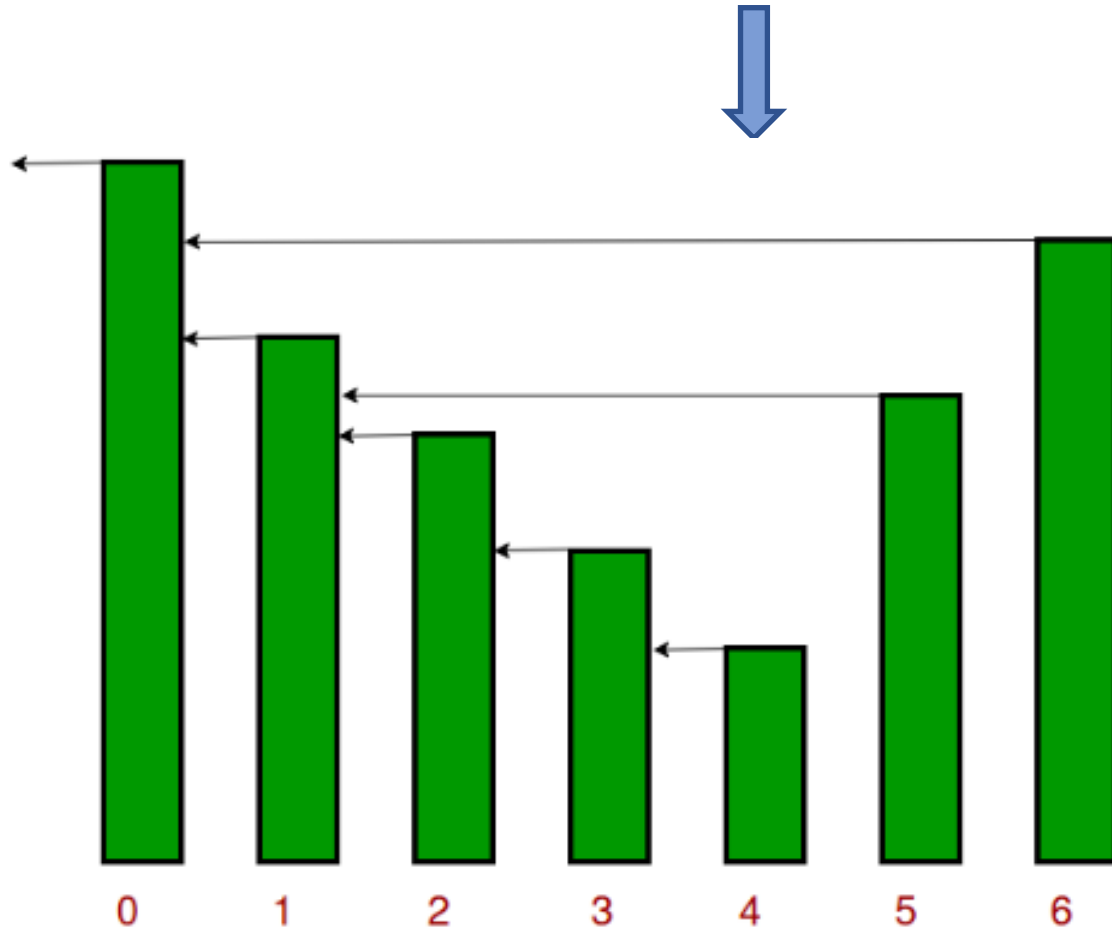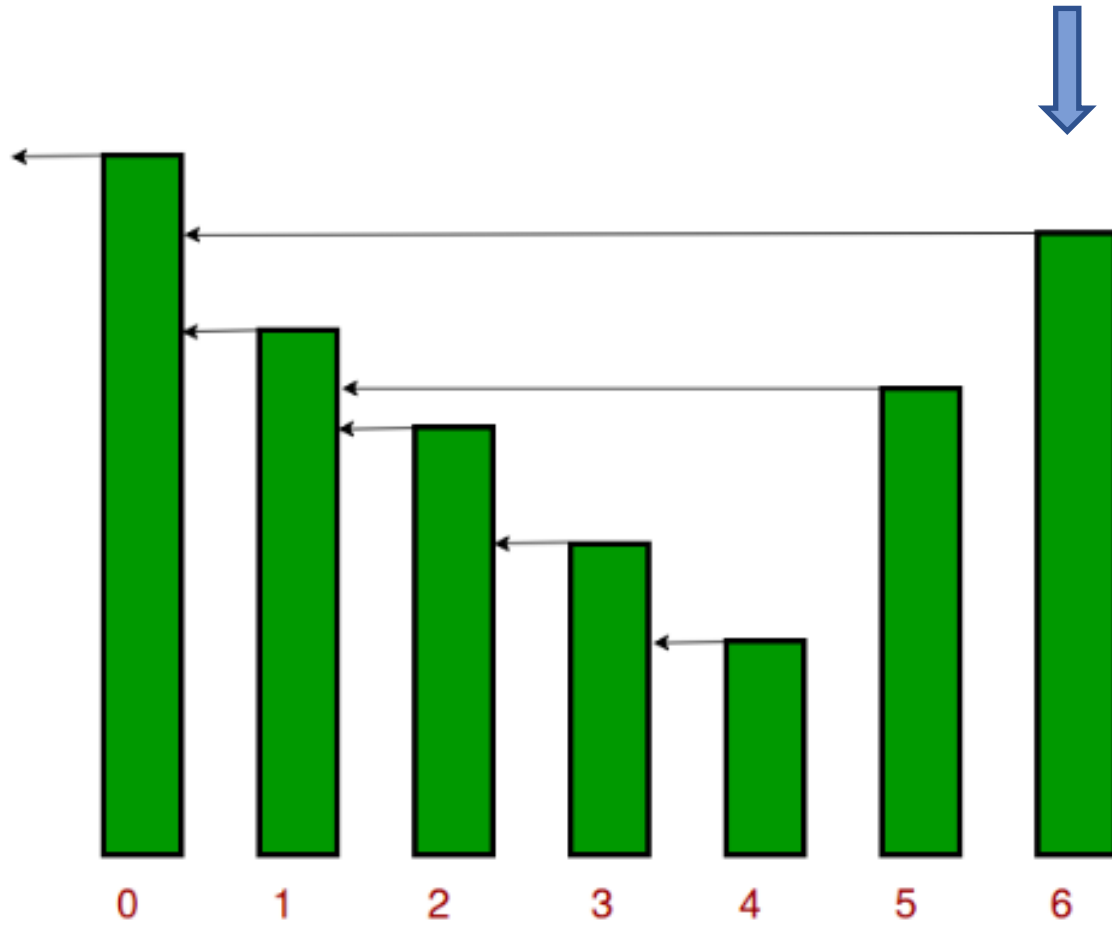
# Class Inheritance

What if there's lots of stuff?

# Class Inheritance in Python

```python
class Location(object):
    def __init__(self, latitude, longitude):
        self.latitude = latitude
        self.longitude = longitude
```

```python
class Stack:
    def __init__(self):
        self.__lst = []
```

## All classes inherit from *object*

Reference: https://docs.python.org/3/tutorial/classes.html#inheritance

# Class Inheritance in Python

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

- Shape → circle/square
- Animal → dog/cat/bird
- Transportation → car/train/bike

Reference: https://docs.python.org/3/tutorial/classes.html#inheritance

# Class Inheritance in Python

```python
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def call(self):
        print(self.name, 'can bark')
```

- Attribute
- Method

```python
class Cat(Animal):
    def __init__(self, name, age, sex):
        super().__init__(name, age)
        self.sex = sex

    def call(self):
        print(self.name, 'barking: meow meow')
```

```python
class Dog(Animal):
    def __init__(self, name, age, sex):
        # python 2.x
        super(Dog, self).__init__(name, age)
        # python 3.x
        # super().__init__(name, age)
        # Animal.__init__(self,name, age)
        self.sex = sex
    def call(self):
        print(self.name, 'barking: woof woof')
```

# Dealing with inherited methods

- Define the class to be inherited
- Then define your later classes that will take traits from the original
- Create parameters (e.g. traits) and functions (aka methods) that belong to your class
- Call the methods based on either referring to the class of the object itself (child) or the original class (parent).
- Test: what happens if we call (cat object).call?

# Python string formatting

```python
name = 'bob'
```

```
'Hello, bob!'
```

```python
# python 2.x
'Hello, %s' % name
```

```python
# python 3.x
'Hello, {}'.format(name)
```

```python
# python 3.6+
f'Hello, {name}!'
```

Reference: https://realpython.com/python-string-formatting/

# Application: Animal class

- Make a method that belongs to the parent class
- Make a method that belongs to only one of the child classes
- Create a cat and a dog
- Call the method for each the cat and dog from the parent class
- Call the method from the child class from each of the child classes (including the one where it isn't defined)

# Upcoming: Thursday

- More on classes
- Walk through SE 4

# Multiple inheritance

- Could you have multiple 'parent' classes?
    - Yes!
    - Python starts from left to right (think: R loading packages in order)
    - If functions have the same name, it starts at the 'most recent' going from child, to parent on left, to parent on the right.

```
Example:
class Parent:
    def method1:
        pass
class Parent2:
    def method1:
        pass
class Child(Parent2, Parent):
    def method:
        pass
```

**If you create an object using the Child class:**
- Where will the *method* method be called from?
- What about *method2*?

**If you create an object using the Parent2 class:**
- Where will the *method* method be called from?
- What about *method2*?

# Multiple inheritance

Example:
class Parent:
 def method1:
   pass
class Parent2:
 def method1:
   pass
class Child(Parent2, Parent):
 def method:
   pass

**If you create an object using the Child class:**
- Where will the *method* method be called from?
- What about *method2*?

**If you create an object using the Parent2 class:**
- Where will the *method* method be called from?
- What about *method2*?

# Time permitting: Divvy

(otherwise, work through on own!)

# Divvy example: exploring inheritance

- So, instead of having a **latitude** and **longitude** attribute, we can define our class to have a **location** attribute that contains a **Location** object.
  - **What does this mean??**

# How is this structured?

- If we follow this approach, adding a method to compute the distance from one station to another becomes very simple:

```python
class DivvyStation(object):

    def __init__(self, stationID, name, latitude, longitude, dpcapacity,
    landmark, online_date):

        self.stationID = stationID
        self.name = name
        self.location = Location(latitude, longitude)
        self.dpcapacity = dpcapacity
        self.landmark = landmark
        self.online_date = online_date

    def distance_to(self, other_station):
        d = self.location.distance_to(other_station.location)
        return d
```

# Variables:

trip_id: A unique integer identifier for the trip.

starttime, and stoptime: The start and end time of the trip.

bikeid: A unique integer identifier for the bike used in this trip.

tripduration: The duration (in seconds) of the trip.

from_station_id and to_station_id: The integer identifiers of the origin and destination stations.

from_station_name and to_station_name: The names of the origin and destination stations.

usertype: This field will be either Customer or Subscriber. A "customer" is a rider who purchased a 24-Hour Pass, and a "subscriber" is a rider who purchased an Annual Membership.

gender: The gender of the rider. This field only has a value when the rider is a subscriber.

birthday: The date of birth of the rider. This field only has a value when the rider is a subscriber.

# NumPy

# What is NumPy

- It is a Python library that provides a multidimensional array object

- NumPy arrays have a **fixed size at creation**, unlike Python lists (which can grow dynamically).

- The elements in a NumPy array usually of the same data type

- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data.

- Just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

https://numpy.org/doc/stable/user/whatisnumpy.html

# NumPy: basics

- NumPy's main object is the homogeneous multidimensional array.
- NumPy's array class is called **ndarray.** It is also known by the alias **array.**
  - Note that **numpy.array** is not the same as the Standard Python Library class **array.array**, which only handles one-dimensional arrays and offers less functionality.

https://numpy.org/doc/stable/user/quickstart.html

# Importing

- Importing: **import** numpy **as** np
- Convention re: how to call and refer to it – why not just call numpy?
- Ex of use: a **=** np**.**arange**(15).**reshape**(3, 5)**