Classes in Python

- A `class` is a combination of functions (they are called as methods) and variables, whose main objective is to provide templates for creating `objects`.
- Keyword `class` is used to create classes followed by the `NameOfTheClass`.

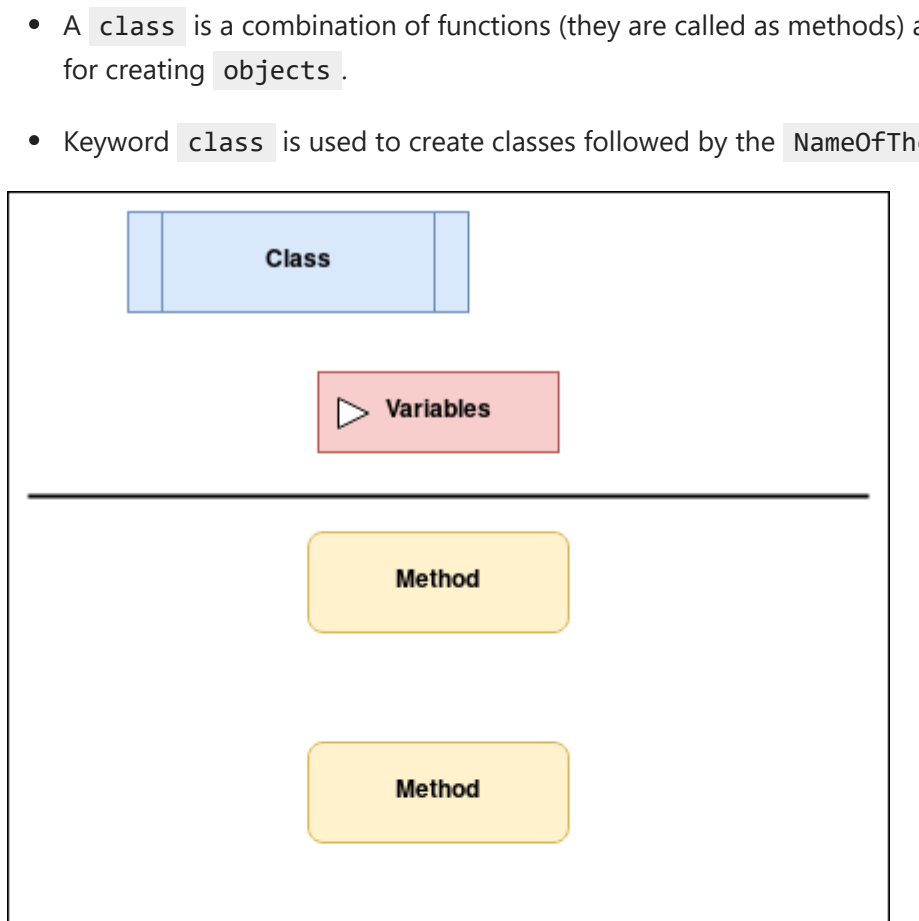


Image by author

Rules to keep in mind

- Name of the class should be `Capital Camel Cased`.
  - `class MyClassName`
  - `class YourClassName`
  - `class OurClassName`
- The methods (name) that are defined in the class should give an essence of verb (i.e., actions).
  - `get_data()`
  - `add()`
  - `organize_response()`

Why Classes?

- The main idea for using class is to implement **Object Oriented Programming (OOP)**.
- To serve or provide template for creating or instantiating specific methods within a program.
- Highly recommended if developping an application in order to avoid any code breaking.
- It provides a way to make our own data type.

Note

- Class is a special data type which defines how to build a certain kind of object.

```
>>> s = "python"
>>> print(type(s))
<class 'str'>
>>> s.upper()
'PYTHON'
```

- `s` is a variable and object.
- The `type(s)` is `str` which is basically a `class`.
- `upper()` is a method / function inside the class.

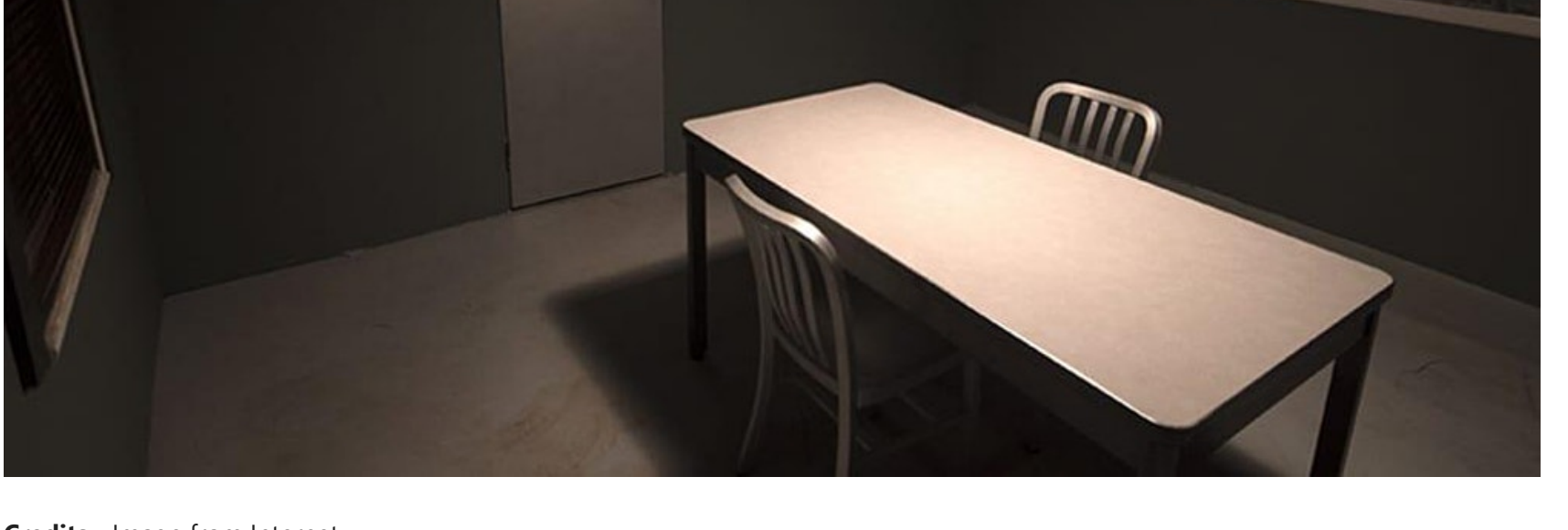
Local variable vs Global variable

Local variable

- Declared inside a function.
- The scope is limited to the particular function alone.
- It is possible to have local variables with the same name in different functions.

Global variable

- Declared outside any function.
- The scope is limited to the entire program file.
- There can be any number of global variables in the program file.



Credits - Image from Internet

```
# show example

some = "python"
print("before - ", some)
print('-----------------')

def my_func():
    print("inside function")
    local = ' coding'
    return "hey " + some + local

print(my_func()) # function call

print('-----------------')
print("unchanged - ", some)

# print("local variable - ", local) # error
```

```
before -  python
-----------------
inside function
hey python coding
-----------------
unchanged -  python
```

Structure of `class`

```
class MyName():
    def __init__(self):
        pass

    def method_1(self, s, w, t):
        # do something
        return None

    def method_2(self):
        # do something

        # method_1 calling
        self.method_1(s, w, t)
        # or
        method_1(self, s, w, t)
        return None
```

Let's make our own data type (class)

```
# class str():
#     def upper(self):
#         return None
#     def lower(self):
#         return None
```

```
s = "python"
print(s.upper())
print(type(s))
```

```
PYTHON
<class 'str'>
```

Create a class called `Myself`

```
# class - Myself
# variable name - "Sameer"

class Myself():
    name = "Sameer"
```

Create an object called `me`

```
me = Myself()
```

Check `type`

```
# dir(s)
```

```
print(type(me))
```

```
<class '__main__.Myself'>
```

Instansiate the variable `name` using the object `me`

```
me.name
```

```
'Sameer'
```

Previously, we created a variable called `s = "python"`. From that, we instantiated the function/method called `upper()`. In the same way, for this example we instantiated the variable called `name` with the object `me`.

Let's take it to another level

Some key rules

- `self` - It is always passed as the first argument in every function or method that is created.

- `__init__()` - A special method which gets initialised without being called. Acts as a `constructor` to initialise the class.
  - Takes `self` as a default argument.
  - No need to have a `return` statement in the method.

- Variables that are declared in `__init__()` method, can be accessed throught the class.
  - These variables are called as `INSTANCE VARIABLES` or `MEMBER VARIABLES`.

- Functions that are defined are called as `INSTANCE METHODS` or `MEMBER METHODS`.
  - Takes `self` as a default argument.
  - Methods are called using the notation `self.<method_name()>`.
  - At the time of calling the method, we need not specify `self` argument.

Note - The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

`__init__()` method → constructor

```
# write code
# make a class HeyPython
# have two print statements inside __init__()

class HeyPython():
    def __init__(self):
        print("Hello")
        print("Bye")
```

No need to call the `__init__()` method

It automatically gets invoked.

```
# create HeyPython object
hpy = HeyPython()
```

```
Hello
Bye
```

`__init__()` will be called automatically when the object is created.

Class with multiple methods

```
# class - Basic
# __init__() - "Hello everyone"
# simple() - "Hey, I am a method, my name is simple()"
# another() -
#     invoke simple()
#     "Hey, I am method, my name is another()"

class Basic():
    def __init__(self):
        print("Hello everyone")

    def simple(self):
        return "Hey, I am a method, my name is simple()"

    def another(self):
        print(self.simple())
        return "Hey, I am method, my name is another()"
```

Object creation

```
b = Basic()
```

```
Hello everyone
```

Check `type`

```
print(type(b))
```

```
<class '__main__.Basic'>
```

Instantiate method

```
b.simple()
```

```
'Hey, I am a method, my name is simple()'
```

```
b.another()
```

```
Hey, I am a method, my name is simple()
'Hey, I am method, my name is another()'
```

Note - There is a lot difference between function and method. They are not the same in terms of `type()` of each.

- For eg:

```
def my_func():
    return True

print(type(my_func))
```

```
<class 'function'>
```

```
class MyClass():
    def simple_func(self):
        return True

cls = MyClass()
print(type(cls.simple_func))
```

```
<class 'method'>
```

Notice the difference. For the first output we got `function` and for the second output we got `method`.

`__init__()` with parameters → parameterized constructors

```
# class - AboutMe()
# params - name, interests, occupation
# method - get_details()
# statement - "The name is {}. My interests are {}. My occupation is {}"

class AboutMe():
    def __init__(self, name, interests, occupation):
        self.name = name
        self.interests = interests
        self.occupation = occupation

    def get_details(self):
        return "The name is {}. My interests are {}. My occupation is {}".format(
            self.name,
            self.interests,
            self.occupation
        )
```

Create an object called `about` without `params`

```
about = AboutMe()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-3d0c9c1ae6ed> in <module>
----> 1 about = AboutMe()

TypeError: __init__() missing 3 required positional arguments: 'name', 'interests', and 'occupation'
```

The above output gives error which is about missing 3 positional arguments. This is because, in our `__init__()` method, we have provided 3 arguments excluding self.

Create an object called `about` with `params`

```
about = AboutMe(
    name='Sameer',
    interests='Coding and Blogging',
    occupation='Mentoring'
)
```

Check `type`

```
print(type(about))
```

```
<class '__main__.AboutMe'>
```

Instansiate the methods using the object `about`

```
about.get_details()
```

```
'The name is Sameer. My interests are Coding and Blogging. My occupation is Mentoring'
```

Templating examples

```
about1 = AboutMe(
    name='Batman',
    interests='Saving people',
    occupation='Vigilante'
)
details1 = about1.get_details()
print(details1)
```

```
The name is Batman. My interests are Saving people. My occupation is Vigilante
```

```
about2 = AboutMe(
    name='Iron Man',
    interests='Making Iron Man suits',
    occupation='Owner of Stark Industries'
)
details2 = about2.get_details()
print(details2)
```

```
The name is Iron Man. My interests are Making Iron Man suits. My occupation is Owner of Stark Industries
```

```
print(type(about))
print(type(about1))
print(type(about2))
```

```
<class '__main__.AboutMe'>
<class '__main__.AboutMe'>
<class '__main__.AboutMe'>
```

```
s = "python"
s1 = "coding"

print(type(s))
print(type(s1))
```

```
<class 'str'>
<class 'str'>
```

```
print(s.upper())
print(s1.upper())
```

```
PYTHON
CODING
```

More detailed Explanation

1. Basic statistics - with classes
2. Phone number shrinking - with classes

1)

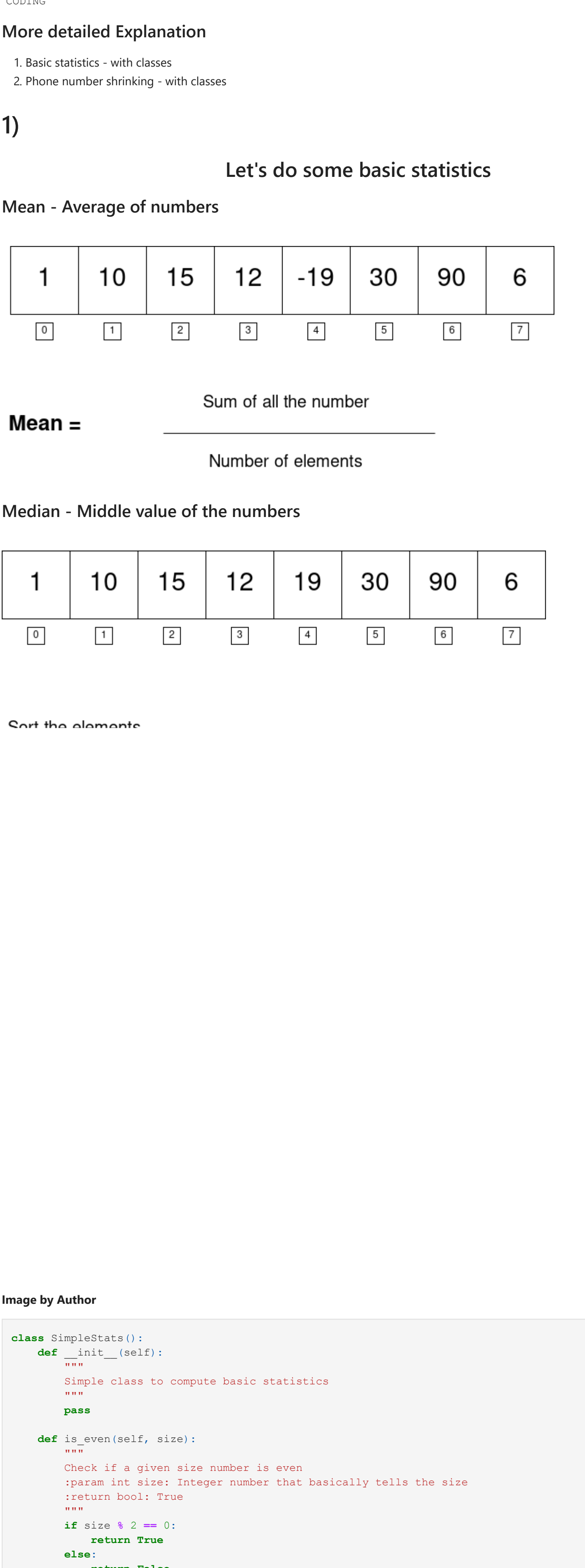


Image by Author

```
class SimpleStats():
    def __init__(self):
        """
        Simple class to compute basic statistics
        """
        pass

    def is_even(self, size):
        """
        Check if a given size number is even
        :param int size: Integer number that basically tells the size
        :return bool: True
        """
        if size % 2 == 0:
            return True
        else:
            return False

    def get_mean(self, array_list):
        """
        Computes the average of the list of numbers
        :param list array_list: List of numbers
        :return float mean: Average value
        """
        sumy = sum(array_list)
        total = len(array_list)
        mean = sumy / total
        return mean

    def get_median(self, array_list):
        """
        Computes the median from the list of numbers
        :param list array_list: List of numbers
        :return any(int, float) median: Median value
        """
        sorted_arrays = sorted(array_list)
        print("The sorted array is: ", sorted_arrays)
        array_size = len(sorted_arrays)

        if self.is_even(array_size):
            med_index_r = array_size // 2
            med_index_l = med_index_r - 1
            sub_array = [sorted_arrays[med_index_l], sorted_arrays[med_index_r]]
            median = self.get_mean(array_list=sub_array)
        else:
            med_index = array_size // 2
            median = sorted_arrays[med_index]

        return median
```

```
stats = SimpleStats()
help(stats)
```

```
Help on SimpleStats in module __main__ object:

class SimpleStats(builtins.object)
 |  Methods defined here:
 |
 |  __init__(self)
 |      Simple class to compute basic statistics
 |
 |  get_mean(self, array_list)
 |      Computes the average of the list of numbers
 |      :param list array_list: List of numbers
 |      :return float mean: Average value
 |
 |  get_median(self, array_list)
 |      Computes the median from the list of numbers
 |      :param list array_list: List of numbers
 |      :return any(int, float) median: Median value
 |
 |  is_even(self, size)
 |      Check if a given size number is even
 |      :param int size: Integer number that basically tells the size
 |      :return bool: True
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
# dir(stats)
```

Object referral

```
stats = SimpleStats()
```

Template making

```
array_list = [1, 5, 10, 32, 44, 53, 76, 9]
mean = stats.get_mean(array_list=array_list)
print("The mean of {} is: {}".format(array_list, mean))
median = stats.get_median(array_list=array_list)
print("The median of {} is: {}".format(array_list, median))
```

```
The mean of [1, 5, 10, 32, 44, 53, 76, 9] is: 28.75
The sorted array is:  [1, 5, 9, 10, 32, 44, 53, 76]
The median of [1, 5, 10, 32, 44, 53, 76, 9] is: 21.0
```

```
array_list = [1, 10, 15, 12, 19, 30, 90, 6]
mean = stats.get_mean(array_list=array_list)
print("The mean of {} is: {}".format(array_list, mean))
median = stats.get_median(array_list=array_list)
print("The median of {} is: {}".format(array_list, median))
```

```
The mean of [1, 10, 15, 12, 19, 30, 90, 6] is: 22.875
The sorted array is:  [1, 6, 10, 12, 15, 19, 30, 90]
The median of [1, 10, 15, 12, 19, 30, 90, 6] is: 13.5
```

```
import random

array_list = [random.choice(range(10, 200)) for i in range(8)]
mean = stats.get_mean(array_list=array_list)
print("The mean of {} is: {}".format(array_list, mean))
median = stats.get_median(array_list=array_list)
print("The median of {} is: {}".format(array_list, median))
```

```
The mean of [148, 108, 120, 118, 38, 35, 136, 80] is: 97.875
The sorted array is:  [35, 38, 80, 108, 118, 120, 136, 148]
The median of [148, 108, 120, 118, 38, 35, 136, 80] is: 113.0
```

2)

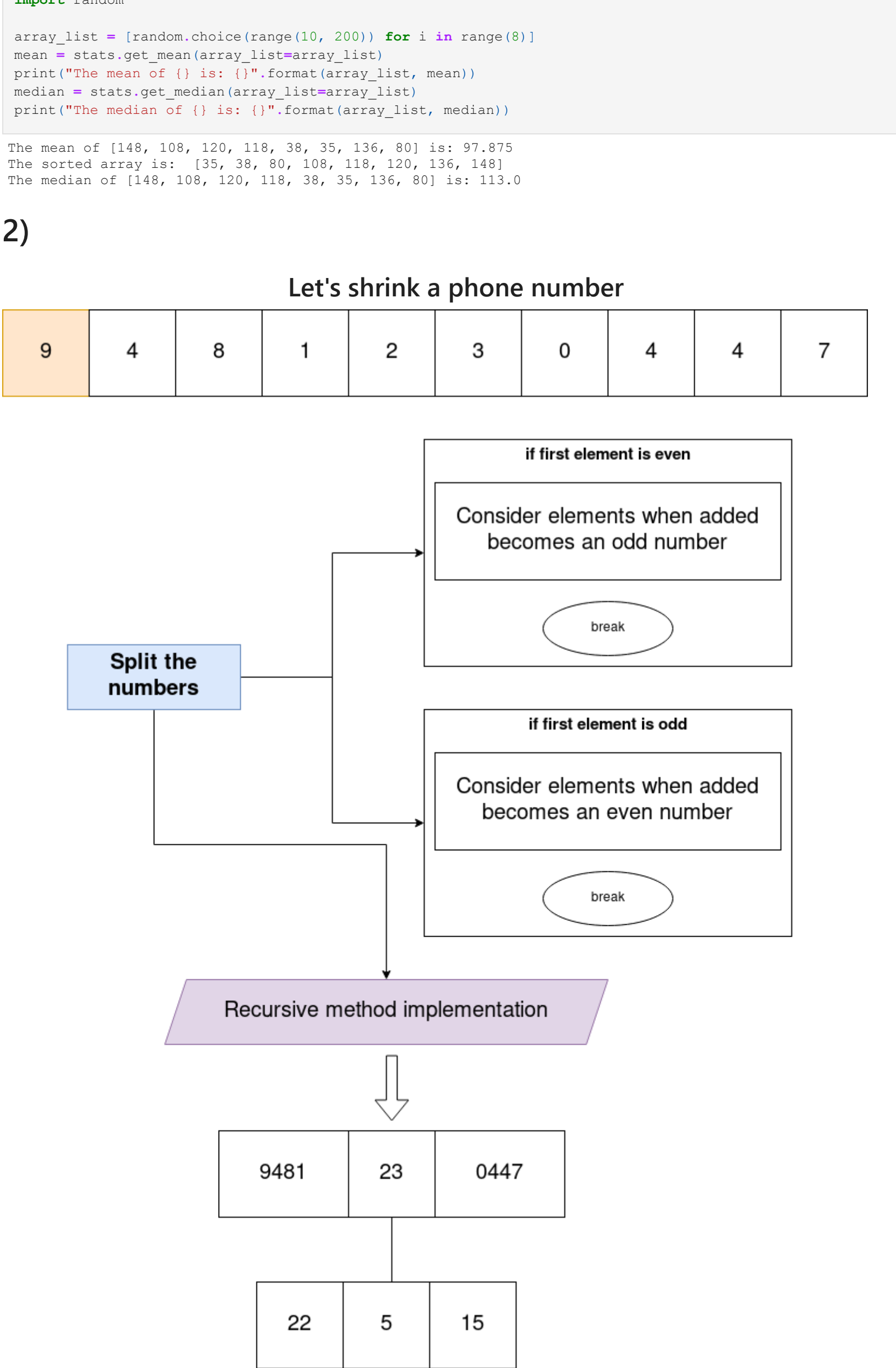


Image by Author

```
class ShrinkNumber():
    def __init__(self, str_num):
        self.str_num = str_num
        print("Original Phone number : {}".format(self.str_num))
        self.total_splits_ = []
        # method calling to get the splits of the number based on the condition
        self.refine_nums()

    def split_nums(self, str_value):
        sum_ = 0
        count_ = 0

        if (int(str_value[0]) % 2 == 0):
            # if the num is even
            for i in range(1, len(str_value) + 1):
                sum_ += int(str_value[i - 1])
                count_ += 1
                if ((sum_ % 2) != 0):
                    self.total_splits_.append(str_value[:count_])
                    # recursive method calling
                    return self.split_nums(str_value[count_:])
        else:
            # if the num is odd
            for i in range(1, len(str_value) + 1):
                sum_ += int(str_value[i - 1])
                count_ += 1
                if ((sum_ % 2) == 0):
                    self.total_splits_.append(str_value[:count_])
                    # recursive method calling
                    return self.split_nums(str_value[count_:])

    def refine_nums(self):
        # check if the length of the number is 10
        if len(self.str_num) == 10:
            e = self.str_num[len(self.str_num) - 1]
            try:
                self.split_nums(str_value=self.str_num)
                self.total_splits_.append(e)
            except Exception as f:
                return "This number is not valid"
        # returning the statement when condition doesn't match
        return "Number exceeds the limit 10"

    def resolve_number(self, num_list):
        manipulated_ = ''
        for i in num_list:
            # summing up each split num after coverting it into int() and reconverting into str()
            manipulated_ += str(sum([int(j) for j in list(i)]))
        return manipulated_

    def shrinked(self):
        if self.total_splits_:
            # method calling to return the shrinked number
            return self.resolve_number(num_list=self.total_splits_)
        else:
            return 'Invalid number'
```

```
phone_number = '9481230447'
new_num = ShrinkNumber(str_num=phone_number)
help(new_num)
```

```
Original Phone number : 9481230447
Help on ShrinkNumber in module __main__ object:

class ShrinkNumber(builtins.object)
 |  ShrinkNumber(str_num)
 |
 |  Methods defined here:
 |
 |  __init__(self, str_num)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  refine_nums(self)
 |
 |  resolve_number(self, num_list)
 |
 |  shrinked(self)
 |
 |  split_nums(self, str_value)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
```

```
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [37]:

```
# dir(new_num)
```

Object referral & Template making

In [38]:

```
phone_number = '9481230447'
new_num = ShrinkNumber(str_num=phone_number)
print("Total splits by odd and even : {}".format(new_num.total_splits_))
print("Shrinked number : {}".format(new_num.shrinked()))
```

```
Original Phone number : 9481230447
Total splits by odd and even : ['9481', '23', '0447']
Shrinked number : 22515
```

In [39]:

```
phone_number = '2124234230'
new_num = ShrinkNumber(str_num=phone_number)
print("Total splits by odd and even : {}".format(new_num.total_splits_))
print("Shrinked number : {}".format(new_num.shrinked()))
```

```
Original Phone number : 2124234230
Total splits by odd and even : ['21', '2423', '423', '0']
Shrinked number : 31190
```

In [40]:

```
phone_number = '9980490439'
new_num = ShrinkNumber(str_num=phone_number)
print("Total splits by odd and even : {}".format(new_num.total_splits_))
print("Shrinked number : {}".format(new_num.shrinked()))
```

```
Original Phone number : 9980490439
Total splits by odd and even : ['99', '8049', '043', '9']
Shrinked number : 182179
```

What did we learn?

- Class definition
- Local variable and Global variable differences
- Parameterized constructor and Non-parameterized constructor
- More detailed explanation of class with examples
  - Basic statistics
  - Phone number shrinking