

Week 11 Tutorial: Classes and Object- oriented Programming

POP77001 Computer Programming for Social Scientists

Recap: Class definition

- Let us recap how a class can be defined in Python.
- Look at the mock class definition below.
- Check that you understand what every part of it is.
- Create an instance of the class.
- Make sure that you can access docstrings for both class and individual methods.

```

1  class MyClass(object):
2      """Docstring describing the class."""
3      # Use docstrings rather than comments to document
4      # both the class and its individual methods
5      # docstrings can be accessed interactively using help()
6      # function. While comments would require going back to
7      # source code.
8
9      # Special Method for object instantiation - always first
10     def __init__(self, var):
11         """Create a new instance of class."""
12         # Methods with two underscores ("__") serve special purposes.
13         # E.g. __init__() is a constructor method that creates
14         # an instance of the class.
15         #
16         # `self` refers to a specific instance of the class.
17         # Then data fields can be described using `self.` prefix
18         self.var = var
19
20     # Special Method for object string representation
21     def __str__(self):
22         """Return a string representation of object."""
23         # Define here how you want an instance of the class to be printed.
24         return str(self.var)
25
26     def class_method(self, arg):
27         """Docstring describing the method."""
28         # Some method that does something.
29         pass

```

Class Definition: Example

- Let us come back to the `StatisticalTest` base class from the lecture.
- Check that you understand what each of the lines of code does.
- Try creating and manipulating the objects of this class.

```

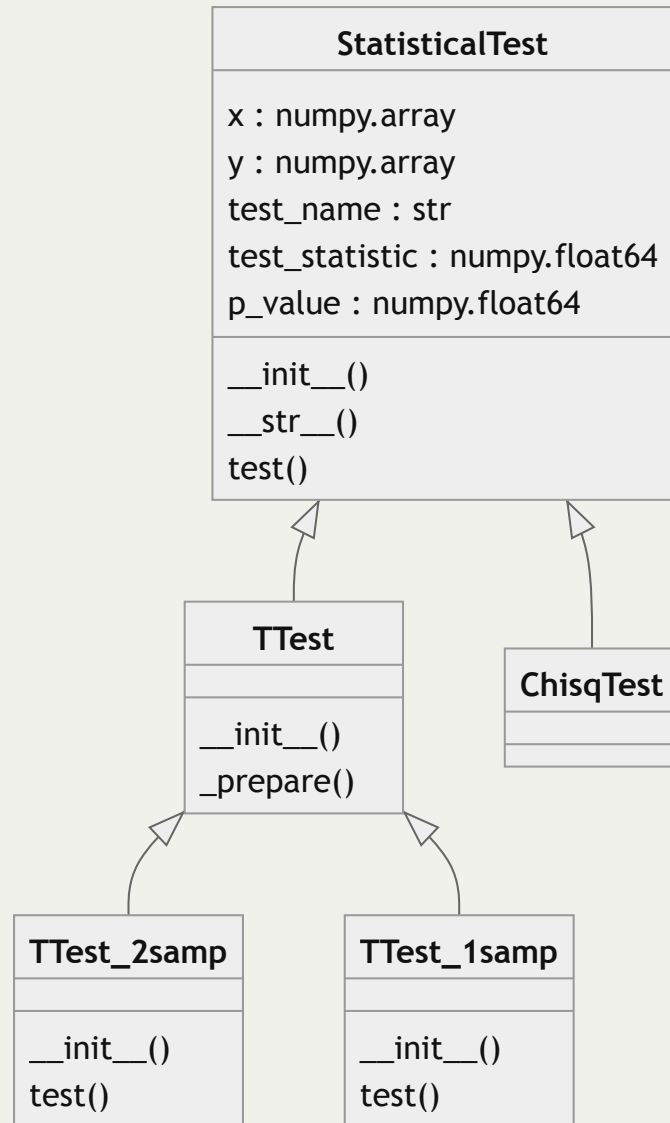
1 class StatisticalTest:
2     """Base class for Statistical tests"""
3     def __init__(self, x, y = None, test_name = None):
4         """
5         Initialize the StatisticalTest.
6
7         Parameters:
8         - x: The first sample for the test.
9         - y: The second sample for tests that compare two samples (default is None).
10        - test_name: The name of the test as string (default is None).
11        """
12        self.x = x
13        self.y = y
14        self.test_name = test_name
15        self.test_statistic = None
16        self.p_value = None
17    def __str__(self):
18        """Return a string representation of the statistical test."""
19        return f'{self.test_name}\n' \
20               f'Test statistic: {self.test_statistic}\n' \
21               f'P-value: {self.p_value}'
22    def test(self):
23        """
24        Conduct the statistical test.
25
26        This method performs the necessary calculations
27        to obtain the test statistic and p-value based
28        on the provided samples.
29
30        Important: This method must be implemented in subclasses.
31        """
32        raise NotImplementedError

```

Exercise: Creating Methods

- Note that there only one method designed to be explicitly called by a user.
- It is the method for conducting a test.
- Note that the results of the test are stored in the object itself.
- We can access them by calling the object's attributes.
- However, in OOP paradigm it is also common to have methods retrieve object attributes for us (see `get_age()` method for `Tamagotchi` class from the lecture).
- Implement a method called `get_results()` that returns the test results as a dictionary.
- Try out a new class design.

StatisticalTest and Subclasses



Exercise: Inheritance Hierarchy

- In the lecture we further implemented classes `TTest` and `TTest_2samp` as subclasses of `StatisticalTest`.
- We sketched a possible `TTest_1samp` class, but have not implemented it.
- Below you will find a function `ttest_1samp()` that performs a one-sample t-test.
- Implement a class `TTest_1samp` that inherits from `TTest` and conducts a one-sample t-test.
- Create an object of this class and conduct a test.
- Extra: compare the results and workflow with the built-in `t.test()` function in R.


```

1 import numpy as np
2 from scipy.stats import t
3
4 def ttest_1samp(x, mean_null = 0):
5     """
6     Perform two-sided one-sample t-test.
7
8     Parameters:
9     - x: The sample for the test.
10    - mean_null: The null hypothesis value (default is 0).
11
12    Returns:
13    - res: A dictionary containing the results of the test.
14    """
15    # Ensure x is NumPy array
16    x = np.array(x)
17    mean_x = np.mean(x)
18    # Use ddof = 1 for sample variance in Python
19    var_x = np.var(x, ddof = 1)
20    n_x = len(x)
21    se_x = np.sqrt(var_x / n_x)
22    # Calculate the t-statistic
23    t_stat = (mean_x - mean_null) / se_x
24    # Calculate the degrees of freedom
25    df = n_x - 1
26    # Calculate the p-value
27    p_val = 2 * (1 - t.cdf(abs(t_stat), df))
28    # Create a result dictionary
29    res = {
30        't_statistic': t_stat,
31        'p_value': p_val
32    }
33    return res

```

```

1 ttest_1samp([-1, 0, 2, 3, 5])

```

```

{'t_statistic': 1.6858544608470492, 'p_value': 0.16710331573259807}

```

Week 11: Assignment 4

- Data Wrangling in Python and Classes
- Due by 12:00 on Monday, 25th November