Usage Guide

Tutorial on how to fill in the spreadsheet and create the class structure templates.

# General

The purpose is to create class core structures to be written automatically from a spreadsheet. It does not replace the actual codes to be inserted on the classes, but rather helps on quickly create the whole software architecture, inheritance, etc. It also creates initial codes for unit tests for each class, checking for variable types, and each class methods.

To be done:

* Add code to create auto plantUML.
* Add the pydoc or sphinx to create auto documentation.

# Syntax

* Each tab on the spreadsheet represents a different class name. Please, use a first capital letter for class names.
* Classes can be inherited from another one by using the syntax on tab name:
  + **Child**Class**#Parent**Class
  + Note:
    - No spaces allowed.
    - The symbol “**#**” means “**inherits**” on that case.
* Example:
  + Class “Animal” as the parent.
  + And class “Dog” as child.



* On each class (tab) there are 7 columns:
  + **variables**:
    - List of all variables for that class.
  + **default**
    - The default value for the variable on the same row.
  + **comment**
    - Write a comment above the line of the variable on the same row.
  + **datatype**
    - Define the datatype for the variable on the same row.
    - Only some types are allowed for the datatypes.
    - This is set up by the tab “all\_config”.
  + **methods**
    - Defines the methods for that class (if there are any methods).
    - Note:
      * There must be always a method “**\_\_init\_\_**”.
      * This is the constructor method.
  + **docstring**
    - This is the description for each method class.
    - Note:
      * The docstring is mandatory, since it helps on readability and code documentation.
  + **argslist**
    - Has the list of all arguments for each method on the same row.
    - Each argument is separated by comma.
    - Note:
      * There can be methods with no arguments.
    - The constructors for child classes are special cases.
      * They might sometimes pass arguments for a parent class.
      * For that, add the parent argument name at the right of the child argument, with the syntax:
        + childArg[parentArg]
      * Example:
        + For a “Dog” class inheriting class “Animal”, use:
        + *dogAge[age], dogName[name]*
        + 🡪 where age and name are the variables on the parent class
        + 🡪 dogAge and dogName are the child class variables

# Auto-generated Python Classes

All Python classes are generated automatically are stored under the “**\_\_auto\_gen\_\_**” folder. Below there is an example of the implementation (**parent** and **child** classes):

**Parent** **class** example (**Animal**):

* Spreadsheet:



* Python code:

class Animal(object):

    def \_\_init\_\_(self, age, name):

        """Constructor"""

        # Stores name

        self.name = "name"

        self.age = age

        pass

    def speak(self):

        """The animal speaks. Has no arguments."""

        pass

* Conversion notes:
  + On the constructor, **age** and **name** are listed on the arguments, as expected.
  + The defined class variables **self.name** and **self.age** store the values on default.
    - **self.age = age** is correct, because it gets the input values on the constructor
    - But **self.name = “name”** might be wrong, since it always gets the string **“name”** instead of the actual input **name** variable.
    - Of course, it is possible to pass a string as the default values.
  + There is a comment above variable **self.name**, as defined on the **comment**.
  + The method **speak** is defined with its **docstring**, but has no code below.
    - The user must add the actual code for each method.

**Child** **class** example (**Dog#Animal**) [Dog inherits Animal]:

* Spreadsheet:



* Python code:

from Animal import Animal

class Dog(Animal):

    def \_\_init\_\_(self, barkSound, dogAge, dogName):

        """Constructor"""

        Animal.\_\_init\_\_(self, age = dogAge, name = dogName)

        # Stores name

        self.name = dogName

        self.age = dogAge

        # Type for that class.

        self.type = "dog"

        # Bark sound.

        self.barkSound = barkSound

        pass

* Conversion notes:
  + The constructor has **barkSound**, **dogAge** and **dogName** that werelisted on the arguments listed, as expected.
  + The parent class is instantiated when calling the parent constructor.
    - **Animal**.\_\_init\_\_(self, **age = dogAge**, **name = dogName**)
    - Notice that **age** and **name** are the variables on the parent constructor class.
    - And **dogAge** and **dogName** are the variables on the child constructor.
    - This was defined on **argslist** by: **dogAge[age]**, **dogName[name]**.
  + The class **variables** get their **default** values, as expected.
    - Notice there is a **type** variable that did not exists on the parent.
    - Also, it gets a **default** string value **“dog”**.
  + Notice all **variables** should respect the correct listed **datatypes**.
  + Finally, on the first line, notice that the parent class was imported correctly.
    - This assumes they are all on the same folder!

# Auto-generated Python Unit Test Classes

In addition to the classes, the code also auto-generates templates for the unit tests.

**Disclaimer**:

* The generated unit tests are only templates.
* They only apply the unit tests for the data types on the columns “datatypes”.
  + Note that the values to be teste must be edited on the codes.
* Additional tests for each method depend on the actual implementation of each function.
  + But the template for each is already there.
* Example for the generated unit tests for the **parent** class (**Animal**):

import unittest

from Animal import Animal

class TestAnimalTypes(unittest.TestCase):

    def setUp(self):

        """ Setup function TestTypes for class Animal """

        self.AnimalObj = Animal(age, name)

        self.name = self.AnimalObj.name

        self.age = self.AnimalObj.age

        pass

    def test\_types(self):

        """ Function to test data types for class Animal """

        self.assertIsInstance(self.name, str)

        self.assertIsInstance(self.age, int)

        pass

class TestSpeak(unittest.TestCase):

    def setUp(self):

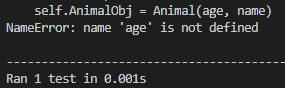
        """ Setup function to test method Animal.speak() """

        pass

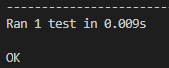
if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

* + It starts by defining the **unittest** module, and **imports** the class to be tested.
    - On that case, the **Animal** class.
  + Then, it starts the unit test class for data types for the variables of the class.
    - class Test**Animal**Types(unittest.TestCase)
    - On that class, the first method is the **setUp**
      * Standard method for the unit test
      * On that method, an object of the target class to be tested is created
      * Example: **self.AnimalObj = Animal(age, name)**
      * Note:
        + Since the variables **age** and **name** are not defined anywhere else (this is just a template), then when executing will flag as error. Unit test result below:



* + - * + Since we know **age** is an integer (**int**) and **name** is a string (**str**), setting, for example, **age=10** and **name=”Pluto”** before the object should do the work. Unit test result below:



* + - * Also, the **setUp** method defines all variables to be tested.
    - The other method on that first class is used for type testing (**test\_types**)
      * It uses the unit test function “**assertIsInstance**”, that checks if the variables defined are actually of defined types.
  + After the first class for data type testing, additional test classes are created for each other method on the original class.
    - For instance, **Animal** has only the “**speak**” method, so only the “**TestSpeak**” class is created:
      * This class does nothing, it only has the standard **setUp** method:
        + Where the user can declare the needed variables, like creating an instance of the object.
      * The user can create additional test classes accordingly:
        + For example, checking the constraints of the returns of the method “**speak**”, using “**test\_speak\_return**”.
        + There are several test functions. Check:

<https://docs.python.org/3/library/unittest.html>

* + At the end, the test class is called by “**unittest.main()**”
  + Just run the code, like “**python test\_Animal.py**” to see the results.
* Below is the example for the generated unit tests for the **child** class (**Dog**):

import unittest

from Dog import Dog

class TestDogTypes(unittest.TestCase):

    def setUp(self):

        """ Setup function TestTypes for class Dog """

        self.DogObj = Dog(barkSound, dogAge, dogName)

        self.name = self.DogObj.name

        self.age = self.DogObj.age

        self.type = self.DogObj.type

        self.barkSound = self.DogObj.barkSound

        pass

    def test\_types(self):

        """ Function to test data types for class Dog """

        self.assertIsInstance(self.name, str)

        self.assertIsInstance(self.age, int)

        self.assertIsInstance(self.type, str)

        self.assertIsInstance(self.barkSound, str)

        pass

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

* + One can note that the code pattern is the same as for the **Animal** class.
    - The main difference are the actual variable names.
    - Also, there is no class for any methods, since the Dog function does not have any methods.
      * It inherits methods from the Animal class, that are not tested here.

How to run?

The procedure to generate the codes is fairly simple. Given that the spreadsheets are configured, browse to where the python code “**pythonClassHierarchy.py**” is, and run it, as:

* **python pythonClassHierarchy**

For the spreadsheet, open “**class\_structure.xlsx**” and make sure that all columns are setup accordingly, for each class:

* Each **tab** is related to a different **class**.
  + Use **child#parent** to inherit from another class.
* Columns **variables**, **default**, **comment** and **datatype** are related to variable setup.
  + Generally, it should have at least one variable.
    - For each variable is a good practice to have a **comment** on its function.
    - It is mandatory to add a **datatype** and a **default** value.
      * The allowed **datatype** is listed on tab **all\_config**
        + Always keep the **all\_config** tab.
        + Add more **allowed\_data\_types** accordingly.
      * The **default** value can be given on the very spreadsheet or can be equal one of the variables listed on **argslist** (for the **\_\_init\_\_**).
* Columns **methods**, **docstrig**, and **argslist** are related to methods setup.
  + It must always have an “**\_\_init\_\_**” methods related to the constructor.
    - On that case the **docstring** can be as simple as “Constructor”
    - Add the input arguments in the **argslist**, separated by **comma**.
  + Additional methods can be added:
    - Is mandatory to add a **docstring** (for documentation purposes).
    - A method can or cannot have arguments.