Pyunitgen

Creating a software application or program is an easy task, howerver a great functional software application is a task that require you to master different level of the SDLC, this, my friend is the haderst task for you as a software developer. On of an area that most software developer lake expertize is the ability to write a `unittest`. A unit testing is another application within your application, is like a `DRY(Don't Repeat Yourself)` principale. But we do that often when it comes to unittesting our code. `unittest` is the driver of any software application, failed to do that, make your software look like a water basket with unseen hole, when looking into the basket, there is no way you can tell whether the basket has a hole or not, but when trying pouring water into it, you will know that the basket is not in the great shape so it is your software application. Some developers want to write the unittest code, but they lake time due to the deadline. Others use manually testing, which in one hand a lot of time consuming, again Others prefer use of `TDD` approach which is double the time of the application delivery. With all that different area , do you think that the user cares ?. To make your life easier, `Pyunitgen`, the first python automatical unittest generator that allow you to focus on your actual features.

Requirement

- `Environment`
  - `Operating System`: GNU/Linux Ubuntu 18.04
- `Software packages`

Packages	Version
python	3.5+

How it works

`Pyunitgen` allows you to tell him, what type data need to be generate for your unittest using `apiDoc` docstring annotation plus some of it's own annotation.

Installation Guide

- `Automatic installation`

  Download the installation script following the below command.

  ```
  ~$ sudo curl -L
  "https://raw.githubusercontent.com/eirtdev/shell/pyunitgen" -o
  /usr/local/bin/pyunitgen && sudo chmod +x /usr/local/bin/pyunitgen
  ```

  Now go ahead and run the below command and wait.

  ```
  ~$ pyunitgen --help
  ```

Getting Start

- `@apiParam`

  ```
  @apiParam [{type=typeOfData}] [field=defaultValue]
  ```

  Describe a parameter passed to your Function/Method.

  Usage: @apiParam {Number} id

Name	Description
`{type}`	Parameter type, e.g. {Boolean}, {Number}, {String}, {Object}, {List} ,{Dict}, ...
`=typeOfData`	The parameters typeOfData e.g =Email, =first_name, =name, =street_address and so on
`[field]`	Fieldname.
`field`	Fieldname with brackets define the Variable as optional.
`=defaultValue`	The parameters default value.

  - `Examples`

    ```
    def ask_first_name(first_name):
        '''
        @apiParam {String} first_name
        '''
        if isinstance(first_name, str):
            return True
        return None
    ```

```
class MyClass:

  def test_unit_with_email(self, test1):
      '''
      @apiParam {String=Email} [test1]

      '''
      pass

  def test_unit_with_default_test2_value_and_string(self, test1,
test2):
      '''
      @apiParam {String=first_Name} [test1]

      @apiParam {String} test2="Another String parameter"
      '''
      pass

  def test_unit_with_number_and_string(self, test1, test2):
      '''
      @apiParam {String=first_Name} [test1]

      @apiParam {Number} test2
      '''
      pass
```

  - `Supported type`

Name	Description
`Number`	

- `@apiReturn`

```
@apiReturn {type} [value]
```

Describe the return value of your Function/Method.

Usage: @apiParam {Number} id

Name	Description
`{type}`	Parameter type, e.g. {Boolean}, {Number}, {String}, {Object}, {List} ,{Dict}, {apiParam}...
`[value]`	The return value of the function.This can be any atomic string , boolean, list, dict , number,class_name in lower case with another method name

  - `Example`

```
class Person:
  name = []

  def set_name(self, user_name):
      '''
      @apiParam {String} [user_name]
      @apiReturn {apiParam.user_name} [person.get_name(0)]
      '''
      self.name.append(user_name)
      return len(self.name) - 1

  def get_name(self, user_id):
      '''
      @apiParam {Number} [user_id=1]
      @apiReturn {String} [There is no such user]
      '''
      if user_id >= len(self.name):
          return 'There is no such user'
      else:
          return self.name[user_id]
```