CSSE1001 Week 5 Practicals

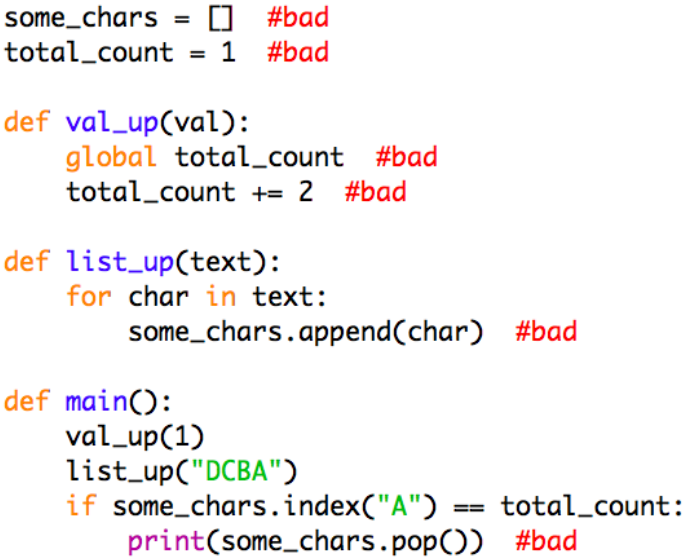
CSSE1001 Course Staff

September 5, 2018

Semester 2, 2018

Some assignment dos and don’ts...

Bad: Global variables

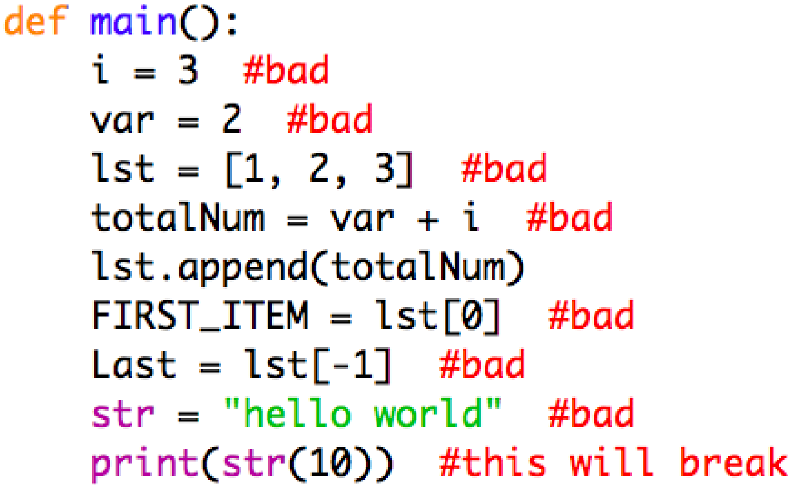


```
some_chars = []  #bad
total_count = 1  #bad

def val_up(val):
    global total_count  #bad
    total_count += 2  #bad

def list_up(text):
    for char in text:
        some_chars.append(char)  #bad

def main():
    val_up(1)
    list_up("DCBA")
    if some_chars.index("A") == total_count:
        print(some_chars.pop())  #bad
```

Good: Use of constants

```python
VAL_INCREMENT = 2
INDEX_CHAR = "A"

def val_up(val):
    val += VAL_INCREMENT
    return val

def list_up(text):
    some_chars = []
    for char in text:
        some_chars.append(char)
    return some_chars

def main():
    value = val_up(1)
    chars = list_up("DCBA")
    if chars.index(INDEX_CHAR) == value:
        print(chars[-1])
```

Bad: Lazy variable naming

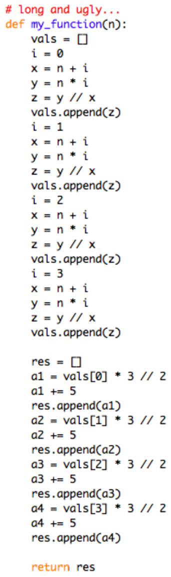


```python
def main():
    i = 3  #bad
    var = 2  #bad
    lst = [1, 2, 3]  #bad
    totalNum = var + i  #bad
    lst.append(totalNum)
    FIRST_ITEM = lst[0]  #bad
    Last = lst[-1]  #bad
    str = "hello world"  #bad
    print(str(10))  #this will break
```

Good: Meaningful variable names

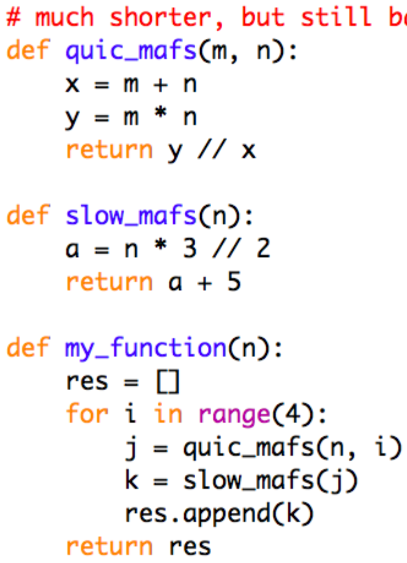
```
def main():
    upper_bound = 3
    lower_bound = 2
    boxes = [1, 2, 3]
    total_num = upper_bound + lower_bound
    boxes.append(total_num)
    first_item = boxes[0]
    last = boxes[-1]
    string = "hello world"
    for i in range(10):  #this is ok
        print(str(i))
```

Bad: Lengthy functions



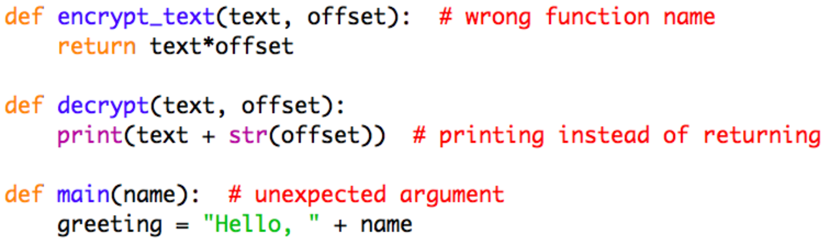
```python
# long and ugly...
def my_function(n):
    vals = []
    i = 0
    x = n + i
    y = n * i
    z = y // x
    vals.append(z)
    i = 1
    x = n + i
    y = n * i
    z = y // x
    vals.append(z)
    i = 2
    x = n + i
    y = n * i
    z = y // x
    vals.append(z)
    i = 3
    x = n + i
    y = n * i
    z = y // x
    vals.append(z)

    res = []
    a1 = vals[0] * 3 // 2
    a1 += 5
    res.append(a1)
    a2 = vals[1] * 3 // 2
    a2 += 5
    res.append(a2)
    a3 = vals[2] * 3 // 2
    a3 += 5
    res.append(a3)
    a4 = vals[3] * 3 // 2
    a4 += 5
    res.append(a4)

    return res
```

Good: Functional decomposition and use of control structures



```python
# much shorter, but still bad variable names
def quic_mafs(m, n):
    x = m + n
    y = m * n
    return y // x

def slow_mafs(n):
    a = n * 3 // 2
    return a + 5

def my_function(n):
    res = []
    for i in range(4):
        j = quic_mafs(n, i)
        k = slow_mafs(j)
        res.append(k)
    return res
```

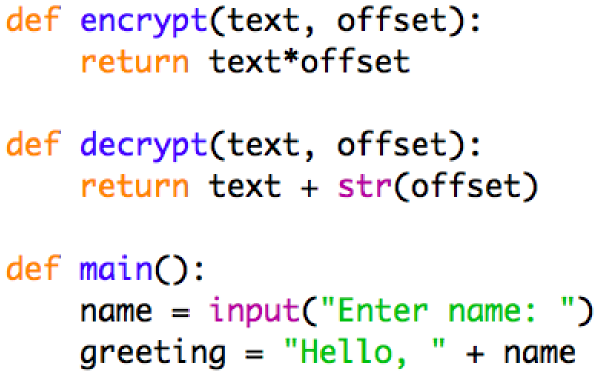
Bad: Not following the spec



```
def encrypt_text(text, offset):  # wrong function name
    return text*offset

def decrypt(text, offset):
    print(text + str(offset))  # printing instead of returning

def main(name):  # unexpected argument
    greeting = "Hello, " + name
```

(note that the implementations are completely absurd...)

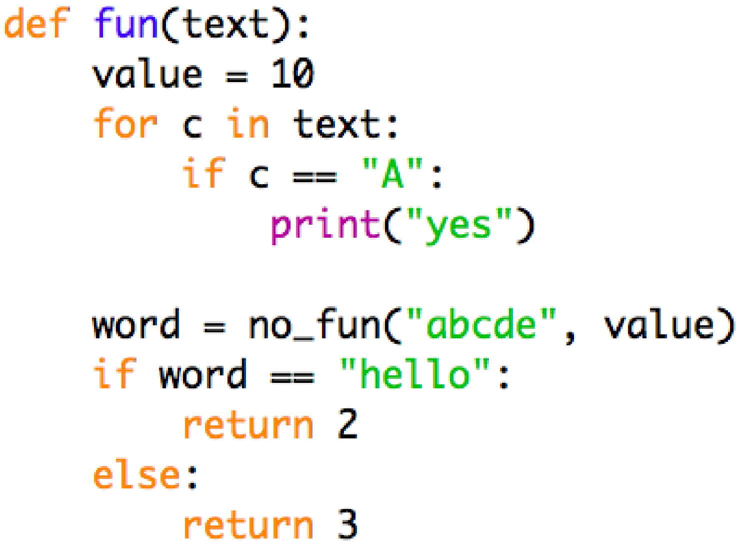
Good: Following the spec exactly



```python
def encrypt(text, offset):
    return text*offset

def decrypt(text, offset):
    return text + str(offset)

def main():
    name = input("Enter name: ")
    greeting = "Hello, " + name
```

(note that the implementations are completely absurd...)

Bad: No documentation

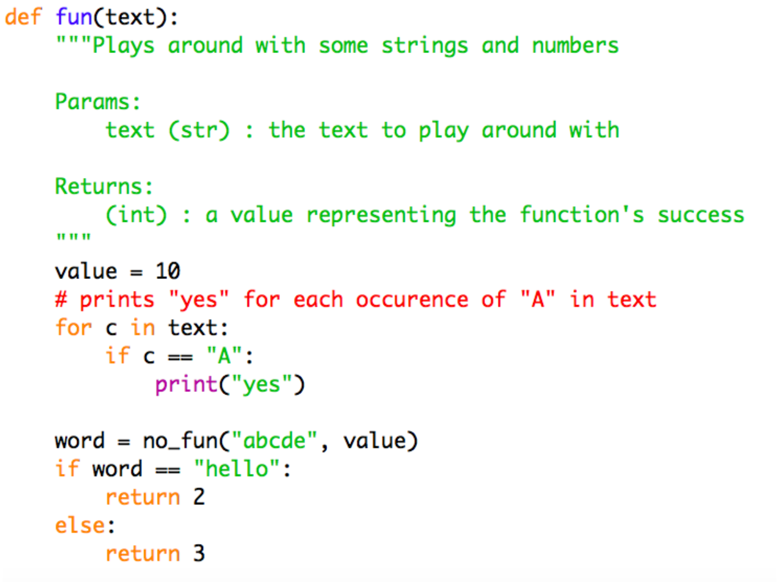


```
def fun(text):
    value = 10
    for c in text:
        if c == "A":
            print("yes")

    word = no_fun("abcde", value)
    if word == "hello":
        return 2
    else:
        return 3
```

Also Bad: Overcommenting

```
def fun(text):
    """Plays around with some strings and numbers
    First the function sets the variable value to 10
    Then it loops over text and prints "yes" when it finds "A"
    Then it calls the function no_fun on the string "abcde"
    If the output of no_fun is "hello" the function returns 2
    If the output of no_fun is not "hello" the function returns 3

    Params:
        text (str) : the text to play around with

    Returns:
        (int) : a value representing the function's success
    """
    value = 10  # sets the variable value to 10
    for c in text:  # loops over each character in text
        if c == "A":  # checks if the character is "A"
            print("yes")  # prints the string "yes" to the screen

    word = no_fun("abcde", value)  # calls the no_fun function
    if word == "hello":  # checks if the word is "hello"
        return 2  # the function outputs the number two
    else:  # if the word is not equal to "hello"
        return 3  # the function outputs the number three
```

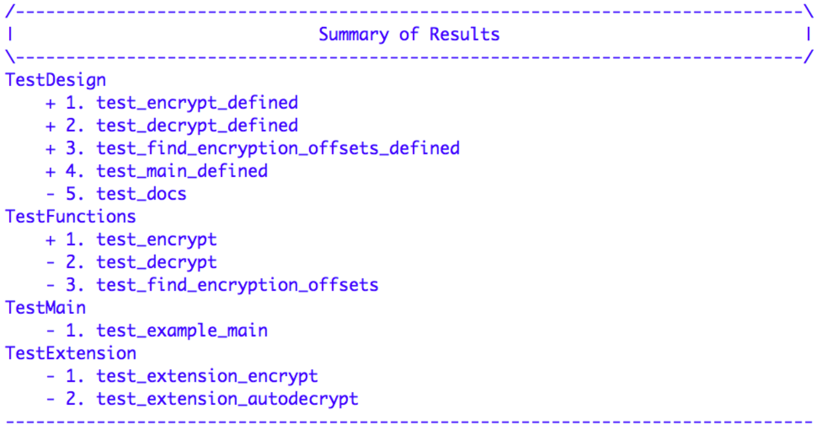
Good: Appropriate docstrings and comments



```python
def fun(text):
    """Plays around with some strings and numbers

    Params:
        text (str) : the text to play around with

    Returns:
        (int) : a value representing the function's success
    """
    value = 10
    # prints "yes" for each occurence of "A" in text
    for c in text:
        if c == "A":
            print("yes")

    word = no_fun("abcde", value)
    if word == "hello":
        return 2
    else:
        return 3
```

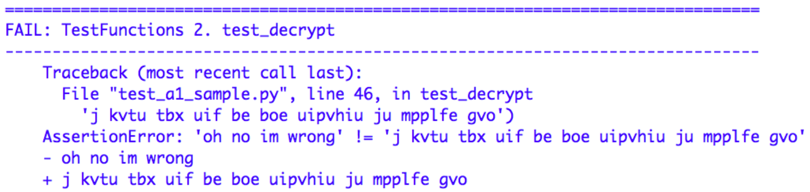
Sample tests

Example output



```
/------------------------------------------------------------------------------\
|                              Summary of Results                              |
\------------------------------------------------------------------------------/
TestDesign
    + 1. test_encrypt_defined
    + 2. test_decrypt_defined
    + 3. test_find_encryption_offsets_defined
    + 4. test_main_defined
    - 5. test_docs
TestFunctions
    + 1. test_encrypt
    - 2. test_decrypt
    - 3. test_find_encryption_offsets
TestMain
    - 1. test_example_main
TestExtension
    - 1. test_extension_encrypt
    - 2. test_extension_autodecrypt
--------------------------------------------------------------------------------
```

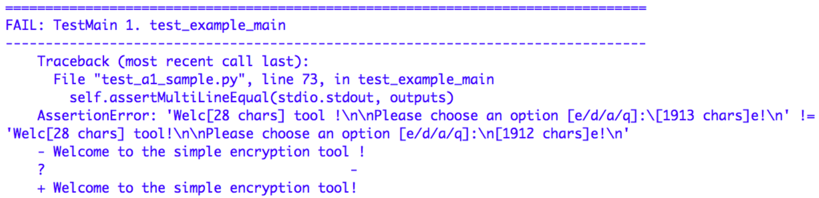
Interpreting failures



```
======================================================================
FAIL: TestFunctions 2. test_decrypt
----------------------------------------------------------------------
   Traceback (most recent call last):
     File "test_a1_sample.py", line 46, in test_decrypt
       'j kvtu tbx uif be boe uipvhiu ju mpplfe gvo')
   AssertionError: 'oh no im wrong' != 'j kvtu tbx uif be boe uipvhiu ju mpplfe gvo'
   - oh no im wrong
   + j kvtu tbx uif be boe uipvhiu ju mpplfe gvo
```

Interpreting failures

Make sure your output is *exact*!



```
================================================================================
FAIL: TestMain 1. test_example_main
--------------------------------------------------------------------------------
    Traceback (most recent call last):
      File "test_a1_sample.py", line 73, in test_example_main
        self.assertMultiLineEqual(stdio.stdout, outputs)
    AssertionError: 'Welc[28 chars] tool !\n\nPlease choose an option [e/d/a/q]:\[1913 chars]e!\n' !=
'Welc[28 chars] tool!\n\nPlease choose an option [e/d/a/q]:\n[1912 chars]e!\n'
    - Welcome to the simple encryption tool !
    ?                                      -
    + Welcome to the simple encryption tool!
```

Important note

Passing these test is **NOT** *a guarantee your code works. It is up to you to test and verify that your code works correctly. We will use more tests than this to mark your assignment and verify functionality. Many test cases have not been included.*

Reminder: Assignment is due this Friday at 8:30pm!