Debugging and Testing

Exercise Answers

1. **[This exercise should be submitted through the Assessments section of Moodle for Tutorial Exercise 5]** Debug the following program:

```
Def factorial(n):
  """Calculate the factorial of the given value and return the result.

    The factorial of n is the product of all positive integers less than or equal to n.

    Arguments:
    n -- A positive integer
  """
  result = 1
  while n != 0:
    n = n - 1
    result = result * n
  return result

# Calculate factorial for the first four integers
for i in range(-1, 5):
  print('Factorial of', i, 'is', factorial(i)
```

2. Add assertions to your answer to 1) that ensure the assumptions made in the code are enforced
3. Write unit tests for the program specified below. You should have a separate test function for each possible behaviour.
   1. Assume the program will be written in a file called *X.py* (so your unit test file should begin with `import X`)
   2. The program will have the following functions:
      - *add(x, y)*: adds *y* to *x* and returns the result
      - *sub(x, y)*: subtracts *y* from *x* and returns the result
      - *mul(x, y)*: multiplies *x* by *y* and returns the result
      - *div(x, y)*: divides *x* by *y* and returns the result
      - *calculatestrangevalue(x, y)*: If *y* is greater than *x*, returns the product of *mul(x, y)* and *add(x, y)*. If *x* is greater than *y*, returns the product of *div(x, y)* and *sub(x, y)*. If *x* and *y* are equal, returns *1*
      - is*valid*positive*integer*input(value): Returns True if the string value can be correctly converted to a positive integer, False otherwise
   3. When run, the program will ask the user to enter two positive integers (and will repeatedly ask until they have done so). It will then display the result of calculate*strange*value with those two integers as arguments and exit

```
import X
assert 3 == X.add(1, 2)
assert 1 == X.sub(3, 2)
assert -1 == X.sub(2, 3)
assert 6 == X.mul(2, 3)
assert 3.5 == X.div(7, 2)
assert 6 == X.calculate_strange_value(1, 2)
assert 2.0 == X.calculate_strange_value(2, 1)
assert 1 == X.calculate_strange_value(2, 2)
assert X.is_valid_positive_integer_input(1)
assert not X.is_valid_positive_integer_input(0)
assert not X.is_valid_positive_integer_input(-1)
assert not X.is_valid_positive_integer_input(1.5)
assert not X.is_valid_positive_integer_input("hello")
```

4. Use TDD to write the program specified by your unit tests in 3). To do so, find your first unit test that fails, then write code to make that test pass. Once the first test is passing, add code to make the second test pass too (i.e. after making changes, test the code is passing the first two tests before continuing). Continue this process (incrementally getting one more test to pass) until all tests pass. Include assertions for the parameters in each of the functions to make sure they are the right type.

```
def add(x, y):
  return x + y

def sub(x, y):
  return x - y

def mul(x, y):
  return x * y

def div(x, y):
  return x / y

def calculate_strange_value(x, y):
  if y > x:
    return mul(x, y) * add(x, y)
  elif x > y:
    return div(x, y) * sub(x, y)
  else:
    return 1

def is_valid_positive_integer_input(value):
  if not isinstance(value, (str, int)):
    return False
  try:
    value = int(value)
  except (TypeError, ValueError):
    return False
  return value > 0

def main():
  x = input("Enter a positive integer: ")
  while not is_valid_positive_integer_input(x):
    x = input("Enter a positive integer: ")
  x = int(x)

  y = input("Enter a positive integer: ")
  while not is_valid_positive_integer_input(y):
    y = input("Enter a positive integer: ")
  y = int(y)

  print(calculate_strange_value(x, y))

if __name__ == "__main__":
  main()
```

Last modified: Thursday, 1 February 2024, 10:45 AM