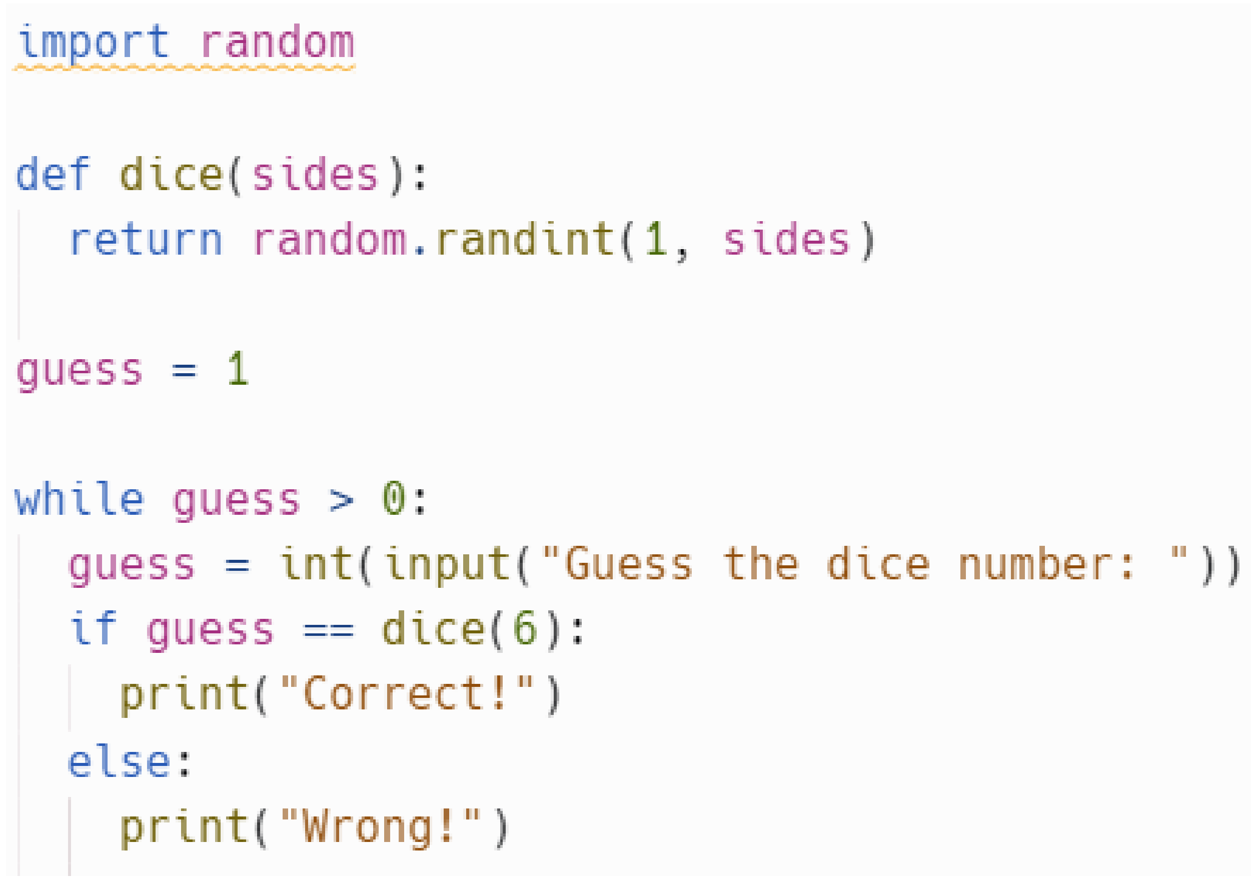
Programming for Social Scientists

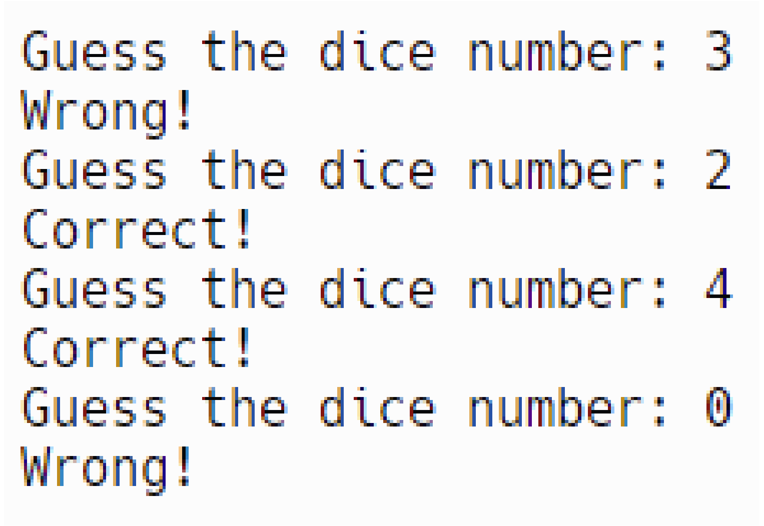
Johan A. Dornschneider-Elkink

Testing, debugging, and exception handling

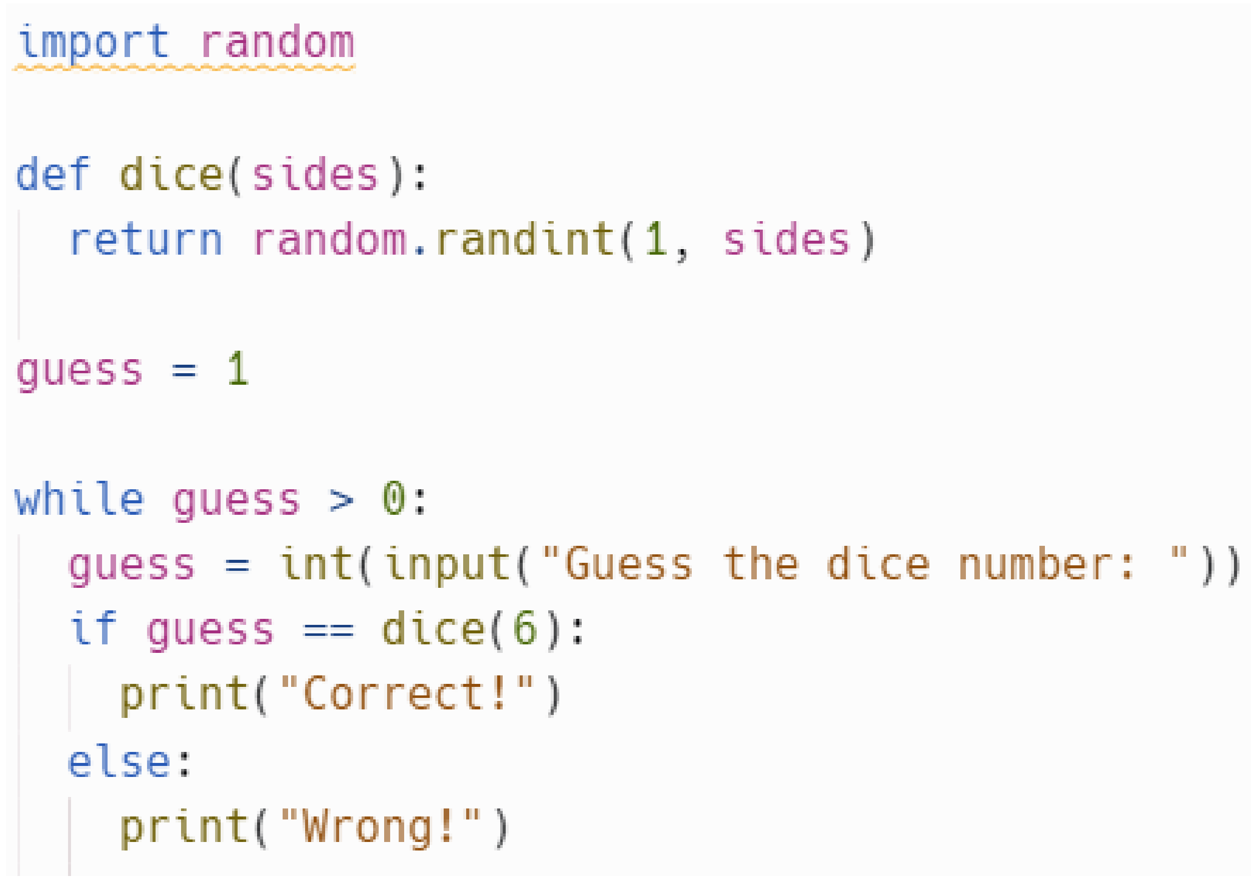
Exception handling



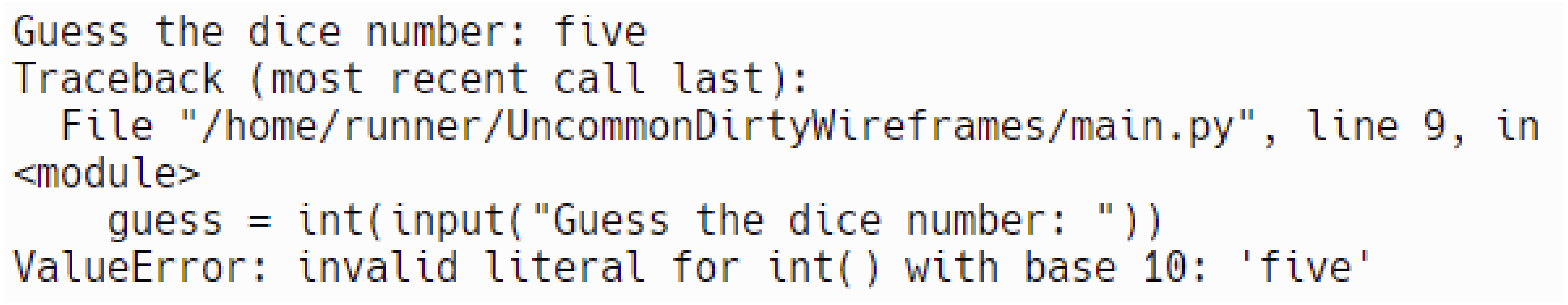
```python
import random

def dice(sides):
  return random.randint(1, sides)

guess = 1

while guess > 0:
  guess = int(input("Guess the dice number: "))
  if guess == dice(6):
    print("Correct!")
  else:
    print("Wrong!")
```



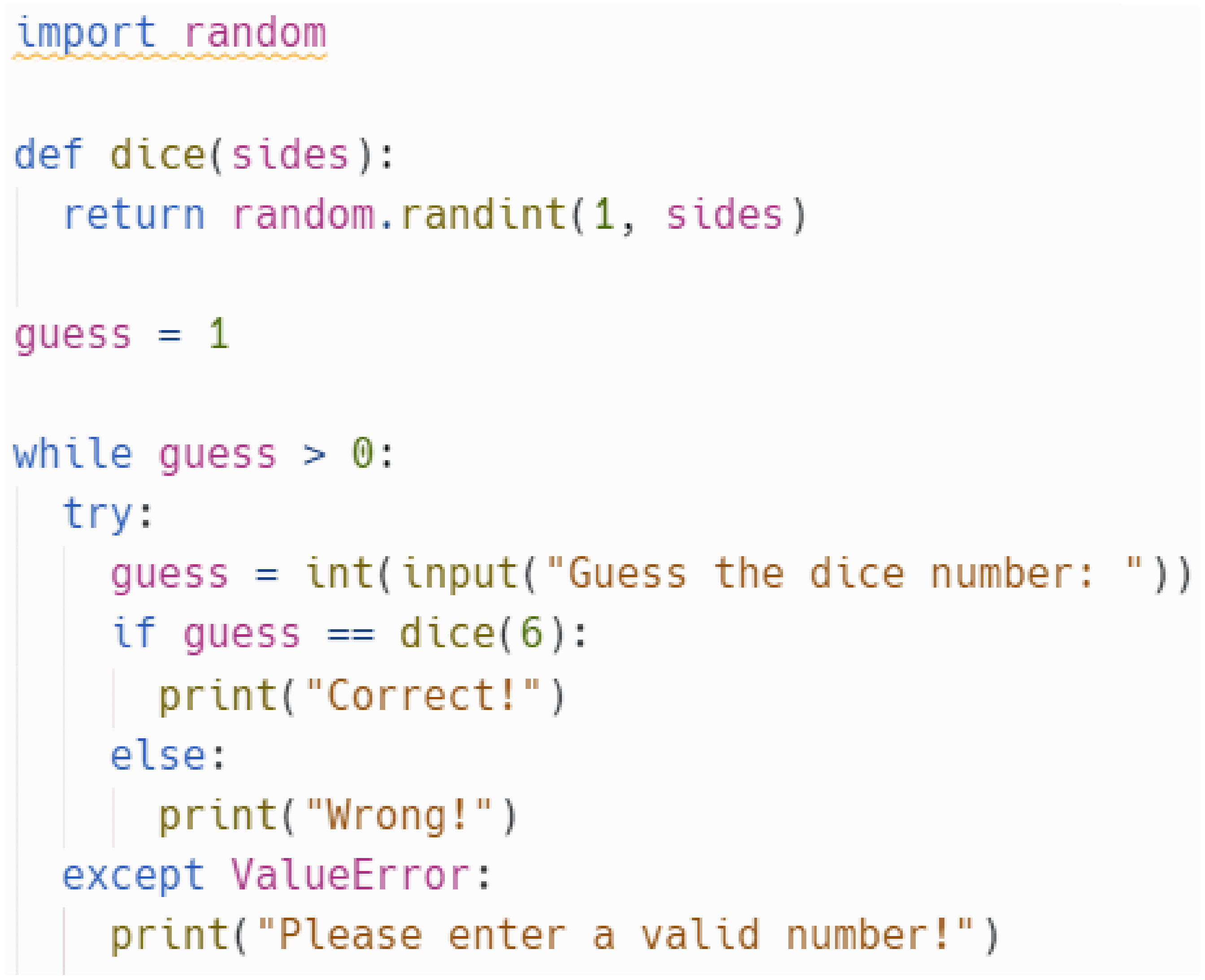
```
Guess the dice number: 3
Wrong!
Guess the dice number: 2
Correct!
Guess the dice number: 4
Correct!
Guess the dice number: 0
Wrong!
```



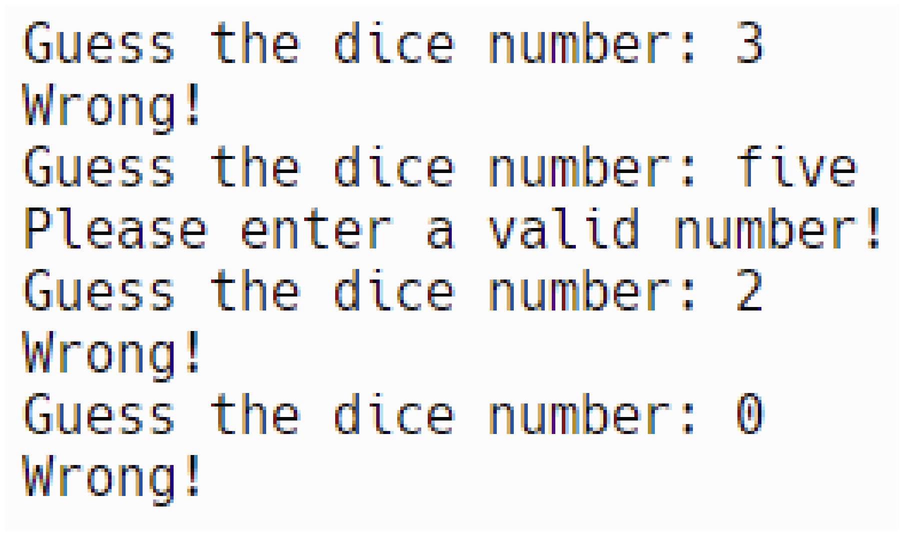
```python
import random

def dice(sides):
  return random.randint(1, sides)

guess = 1

while guess > 0:
  guess = int(input("Guess the dice number: "))
  if guess == dice(6):
    print("Correct!")
  else:
    print("Wrong!")
```



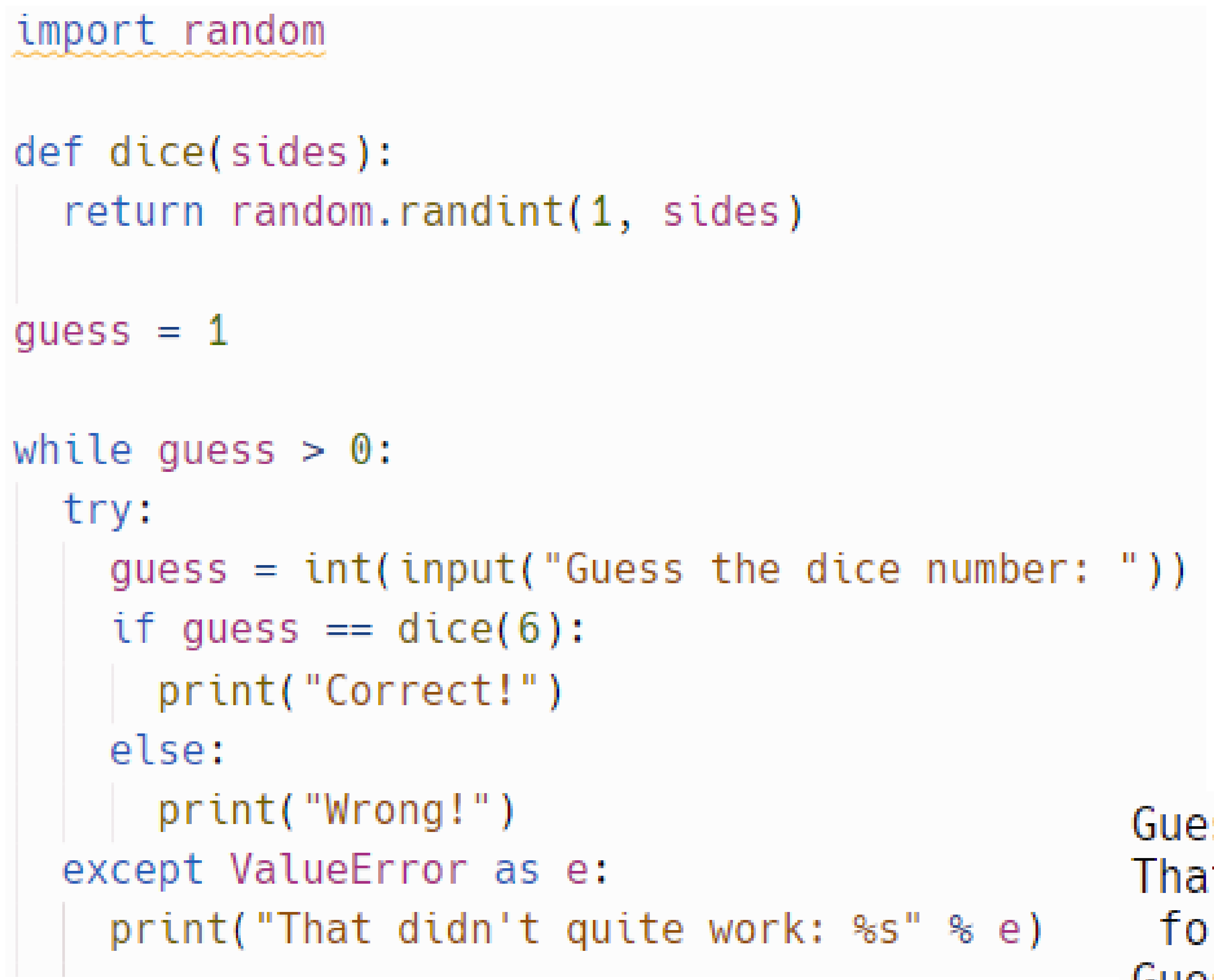
```
Guess the dice number: five
Traceback (most recent call last):
  File "/home/runner/UncommonDirtyWireframes/main.py", line 9, in
<module>
    guess = int(input("Guess the dice number: "))
ValueError: invalid literal for int() with base 10: 'five'
```



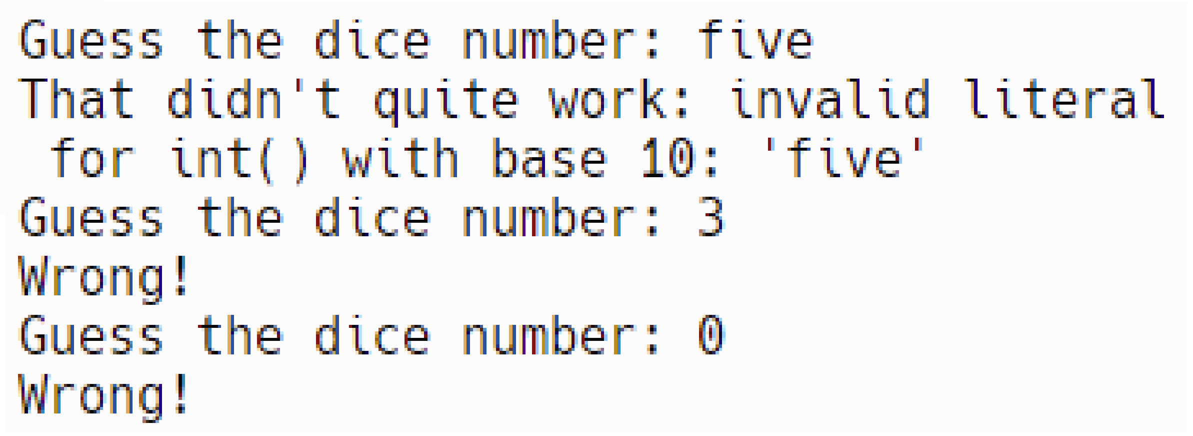
```python
import random

def dice(sides):
  return random.randint(1, sides)

guess = 1

while guess > 0:
  try:
    guess = int(input("Guess the dice number: "))
    if guess == dice(6):
      print("Correct!")
    else:
      print("Wrong!")
  except ValueError:
    print("Please enter a valid number!")
```



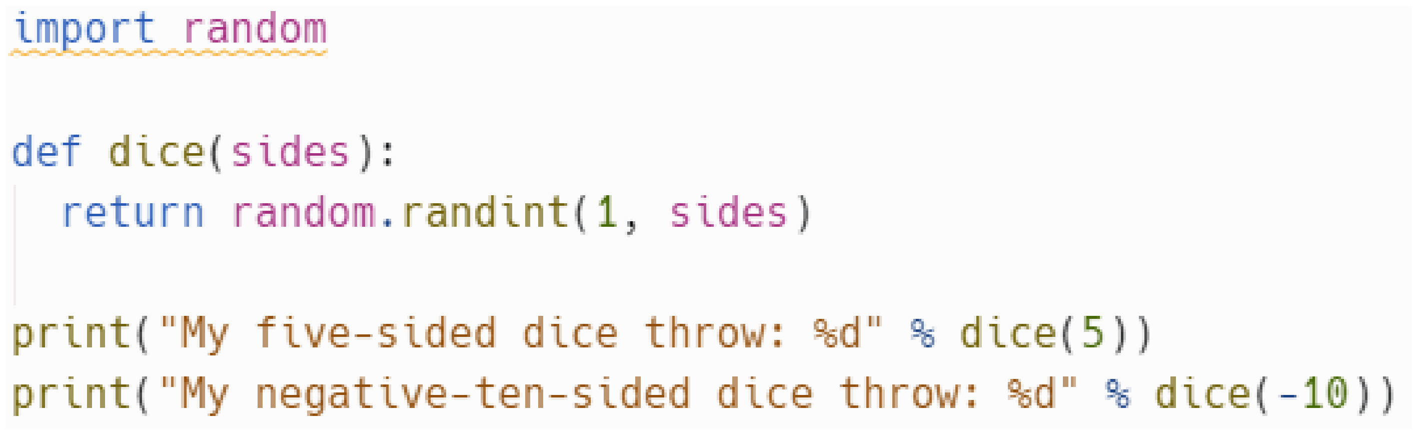
```
Guess the dice number: 3
Wrong!
Guess the dice number: five
Please enter a valid number!
Guess the dice number: 2
Wrong!
Guess the dice number: 0
Wrong!
```



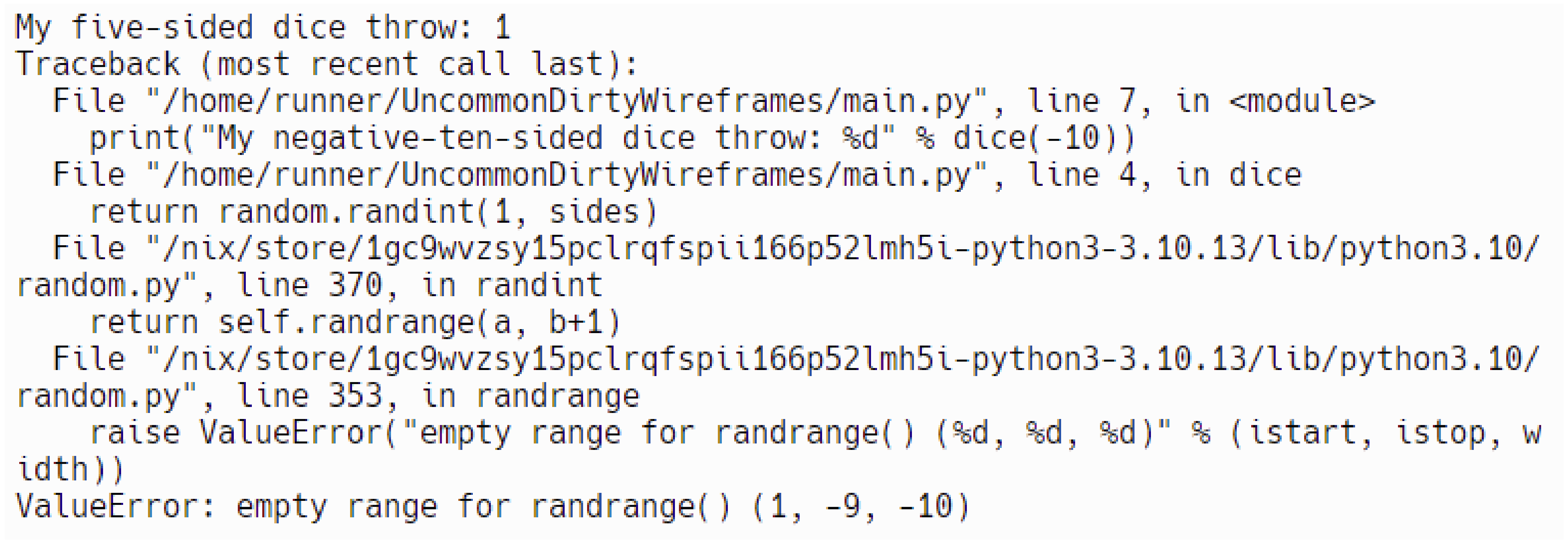
```python
import random

def dice(sides):
  return random.randint(1, sides)

guess = 1

while guess > 0:
  try:
    guess = int(input("Guess the dice number: "))
    if guess == dice(6):
      print("Correct!")
    else:
      print("Wrong!")
  except ValueError as e:
    print("That didn't quite work: %s" % e)
```



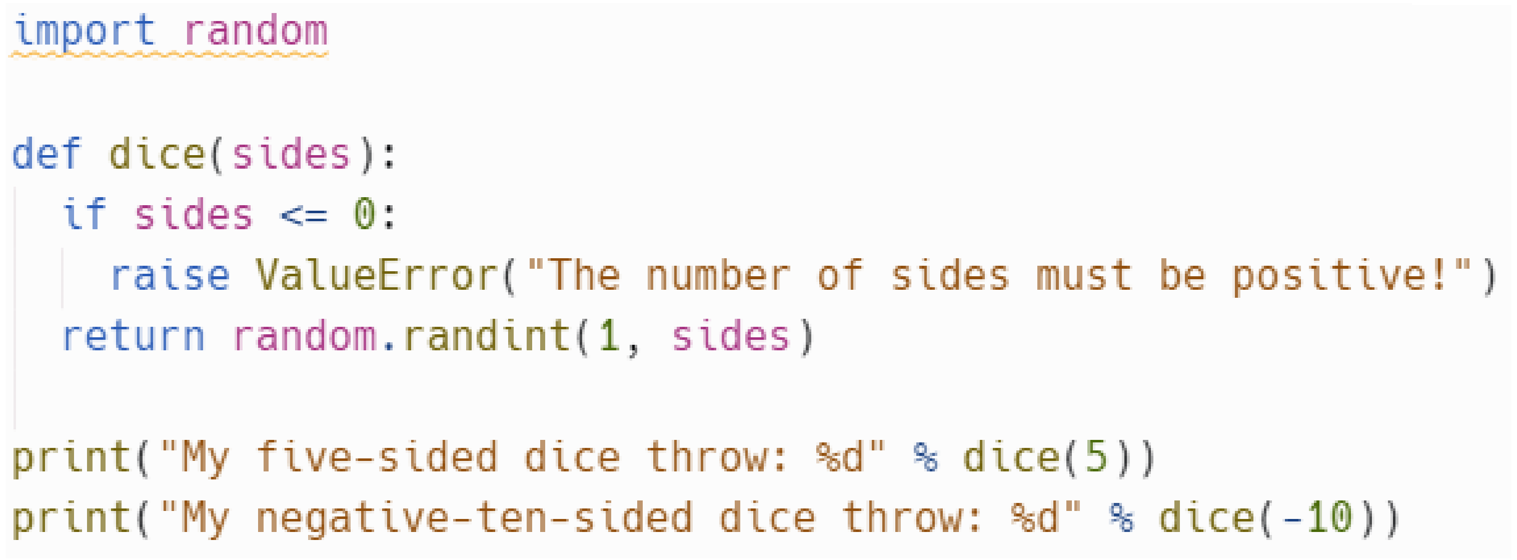
```
Guess the dice number: five
That didn't quite work: invalid literal
 for int() with base 10: 'five'
Guess the dice number: 3
Wrong!
Guess the dice number: 0
Wrong!
```



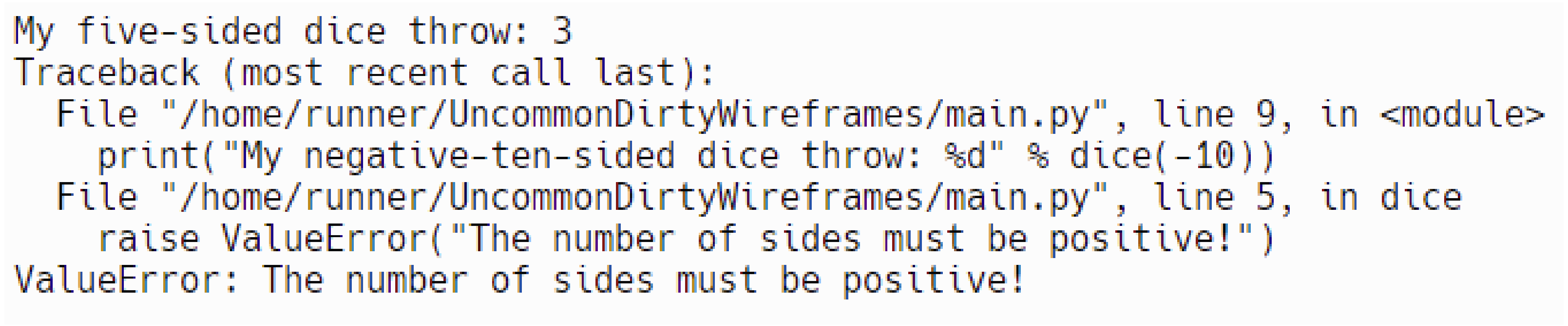
```python
import random

def dice(sides):
  return random.randint(1, sides)

print("My five-sided dice throw: %d" % dice(5))
print("My negative-ten-sided dice throw: %d" % dice(-10))
```



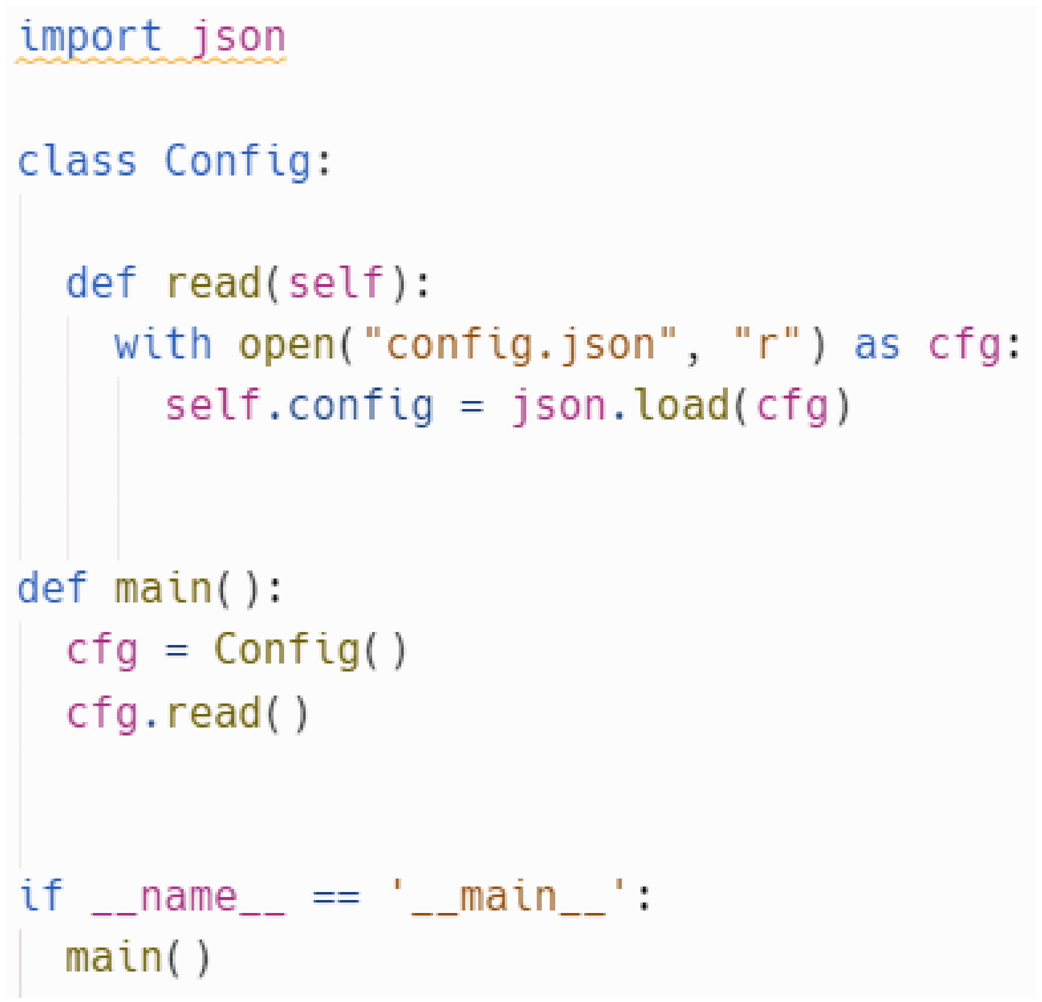
```
My five-sided dice throw: 1
Traceback (most recent call last):
  File "/home/runner/UncommonDirtyWireframes/main.py", line 7, in <module>
    print("My negative-ten-sided dice throw: %d" % dice(-10))
  File "/home/runner/UncommonDirtyWireframes/main.py", line 4, in dice
    return random.randint(1, sides)
  File "/nix/store/1gc9wvzsy15pclrqfspii166p52lmh5i-python3-3.10.13/lib/python3.10/
random.py", line 370, in randint
    return self.randrange(a, b+1)
  File "/nix/store/1gc9wvzsy15pclrqfspii166p52lmh5i-python3-3.10.13/lib/python3.10/
random.py", line 353, in randrange
    raise ValueError("empty range for randrange() (%d, %d, %d)" % (istart, istop, w
idth))
ValueError: empty range for randrange() (1, -9, -10)
```



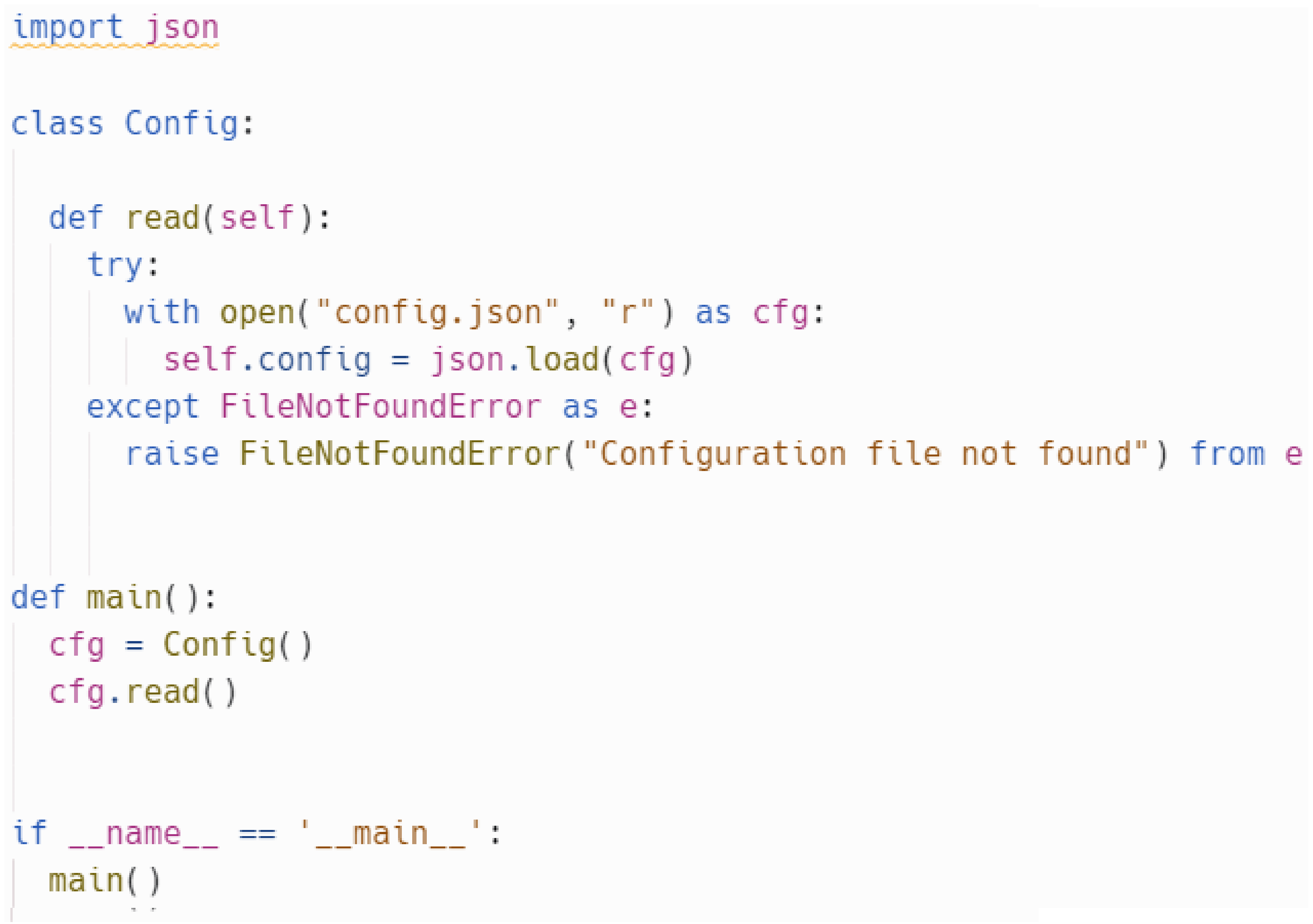
```python
import random

def dice(sides):
  if sides <= 0:
    raise ValueError("The number of sides must be positive!")
  return random.randint(1, sides)

print("My five-sided dice throw: %d" % dice(5))
print("My negative-ten-sided dice throw: %d" % dice(-10))
```



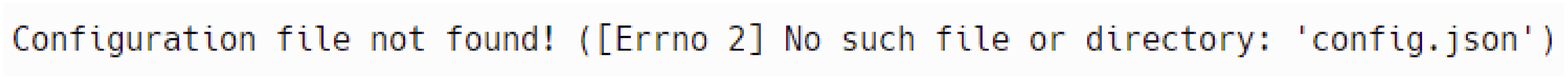
```
My five-sided dice throw: 3
Traceback (most recent call last):
  File "/home/runner/UncommonDirtyWireframes/main.py", line 9, in <module>
    print("My negative-ten-sided dice throw: %d" % dice(-10))
  File "/home/runner/UncommonDirtyWireframes/main.py", line 5, in dice
    raise ValueError("The number of sides must be positive!")
ValueError: The number of sides must be positive!
```



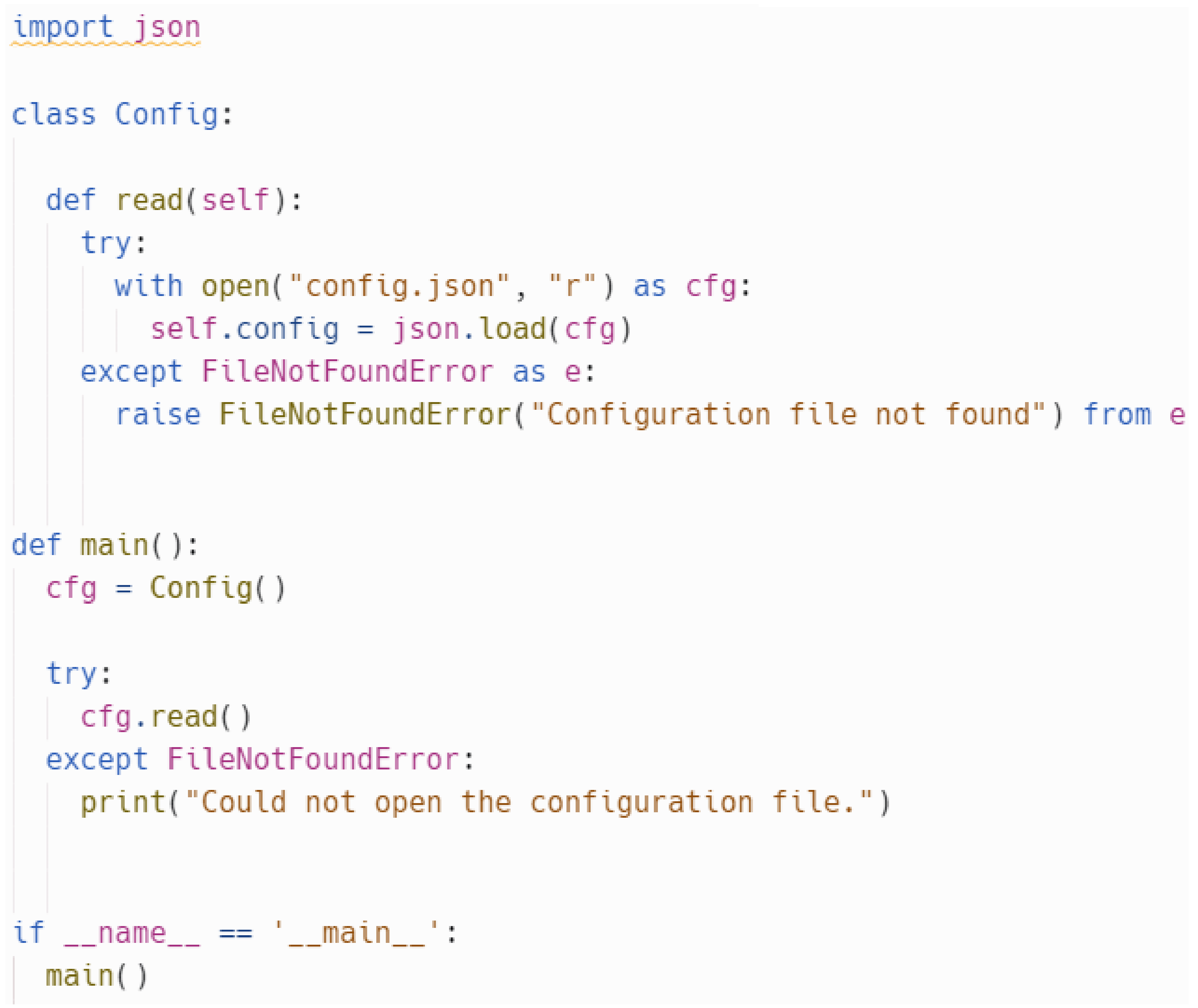
```python
import json

class Config:

  def read(self):
    with open("config.json", "r") as cfg:
      self.config = json.load(cfg)


def main():
  cfg = Config()
  cfg.read()


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



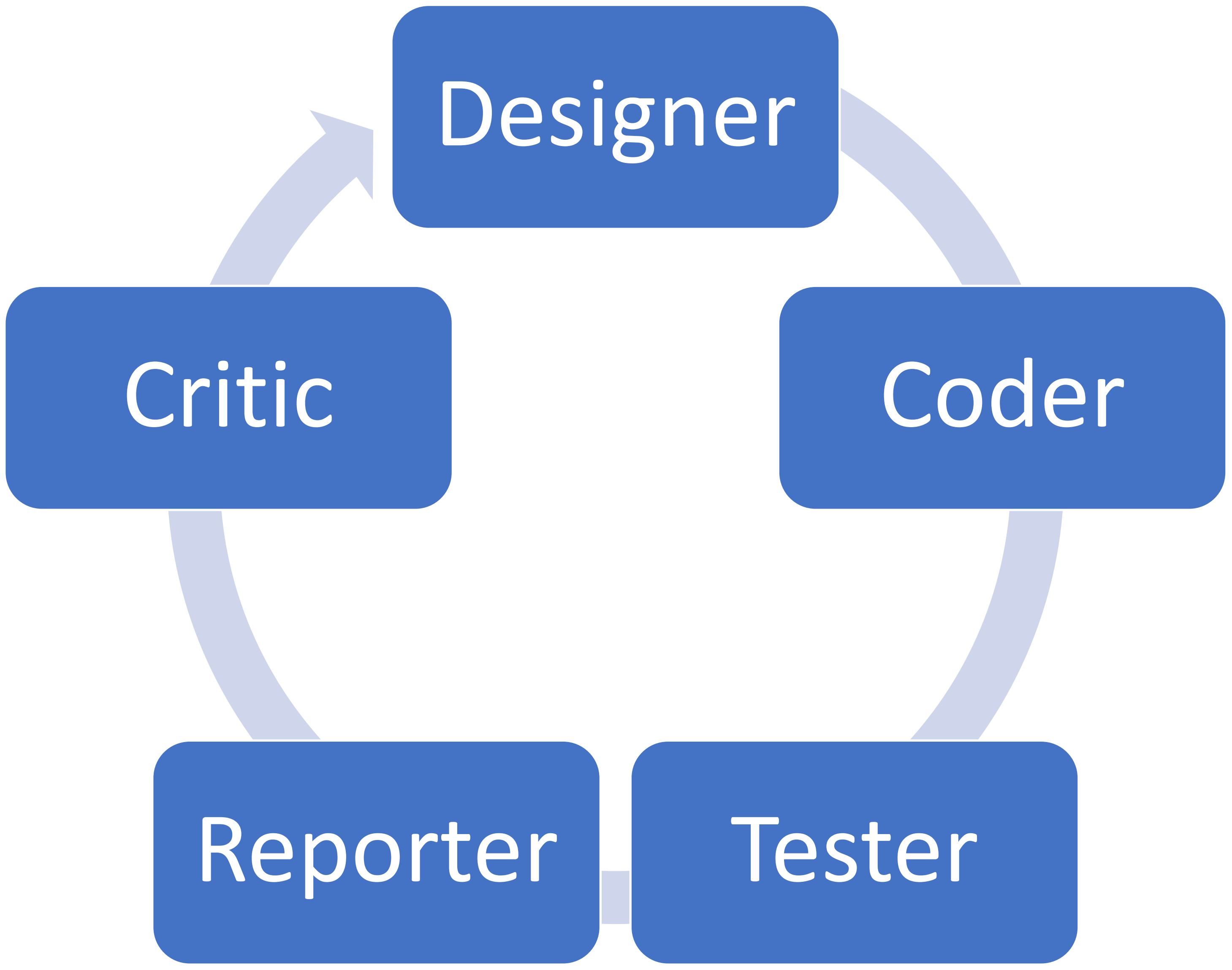
```python
import json

class Config:

  def read(self):
    try:
      with open("config.json", "r") as cfg:
        self.config = json.load(cfg)
    except FileNotFoundError as e:
      raise FileNotFoundError("Configuration file not found") from e


def main():
  cfg = Config()
  cfg.read()


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



```
Configuration file not found! ([Errno 2] No such file or directory: 'config.json')
```



```python
import json

class Config:

  def read(self):
    try:
      with open("config.json", "r") as cfg:
        self.config = json.load(cfg)
    except FileNotFoundError as e:
      raise FileNotFoundError("Configuration file not found") from e


def main():
  cfg = Config()

  try:
    cfg.read()
  except FileNotFoundError:
    print("Could not open the configuration file.")


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



If there is more than one critic, only one moves on to designer today.

```
class Agent:

  def __init__(self, ideology):
    self.ideology = ideology
    self.influence = random.random()

  def setInfluence(self, influence):
    self.influence = influence
```

12 minutes

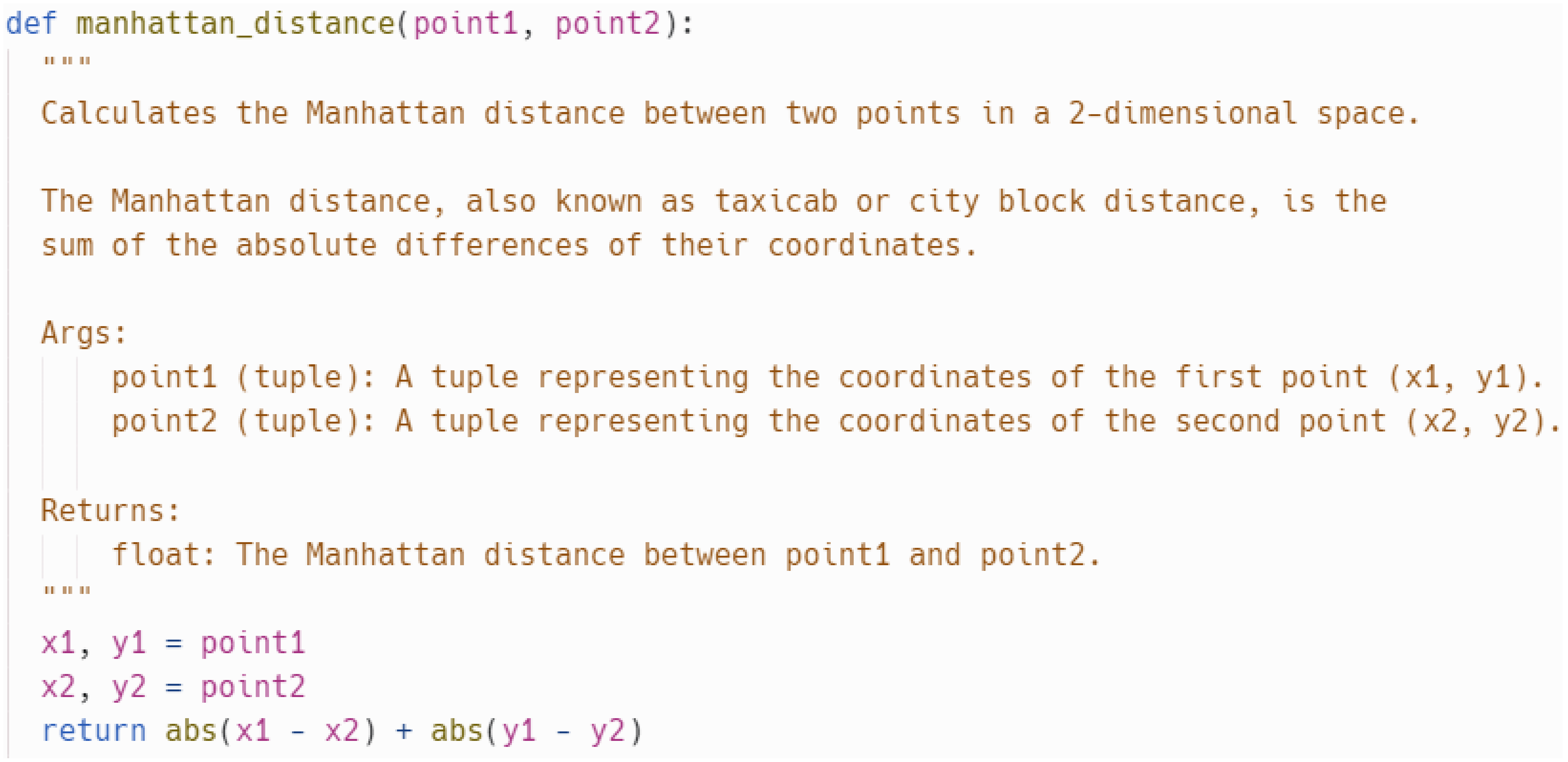
9 x 1 minute reports

TASK 1

Update the above code such that the setInfluence() method throws an appropriate error when influence is not a valid value.

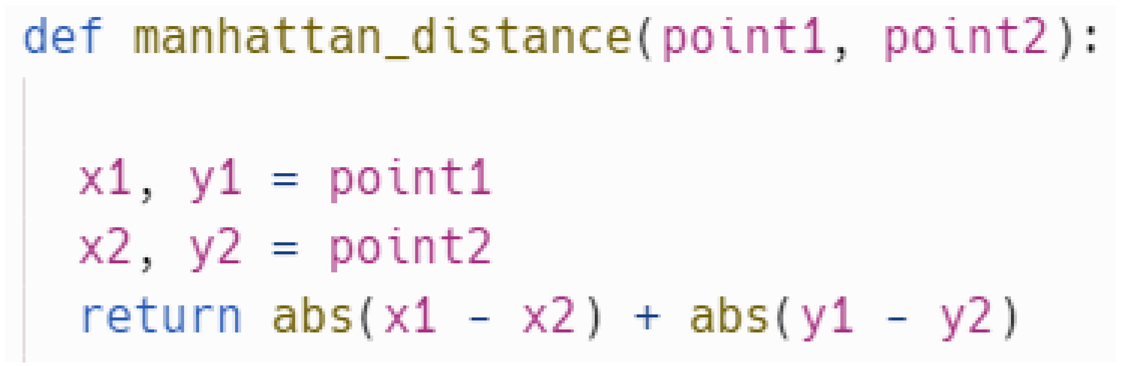
Unit testing

distance.py



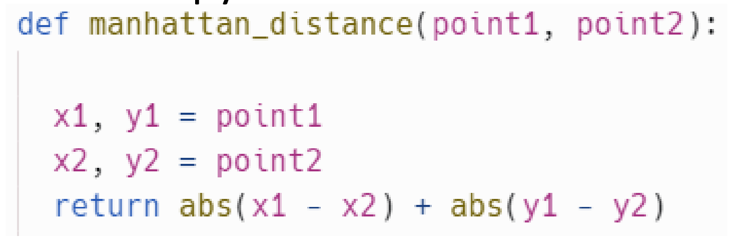
```python
def manhattan_distance(point1, point2):
    """
    Calculates the Manhattan distance between two points in a 2-dimensional space.

    The Manhattan distance, also known as taxicab or city block distance, is the
    sum of the absolute differences of their coordinates.

    Args:
        point1 (tuple): A tuple representing the coordinates of the first point (x1, y1).
        point2 (tuple): A tuple representing the coordinates of the second point (x2, y2).

    Returns:
        float: The Manhattan distance between point1 and point2.
    """
    x1, y1 = point1
    x2, y2 = point2
    return abs(x1 - x2) + abs(y1 - y2)
```

distance.py



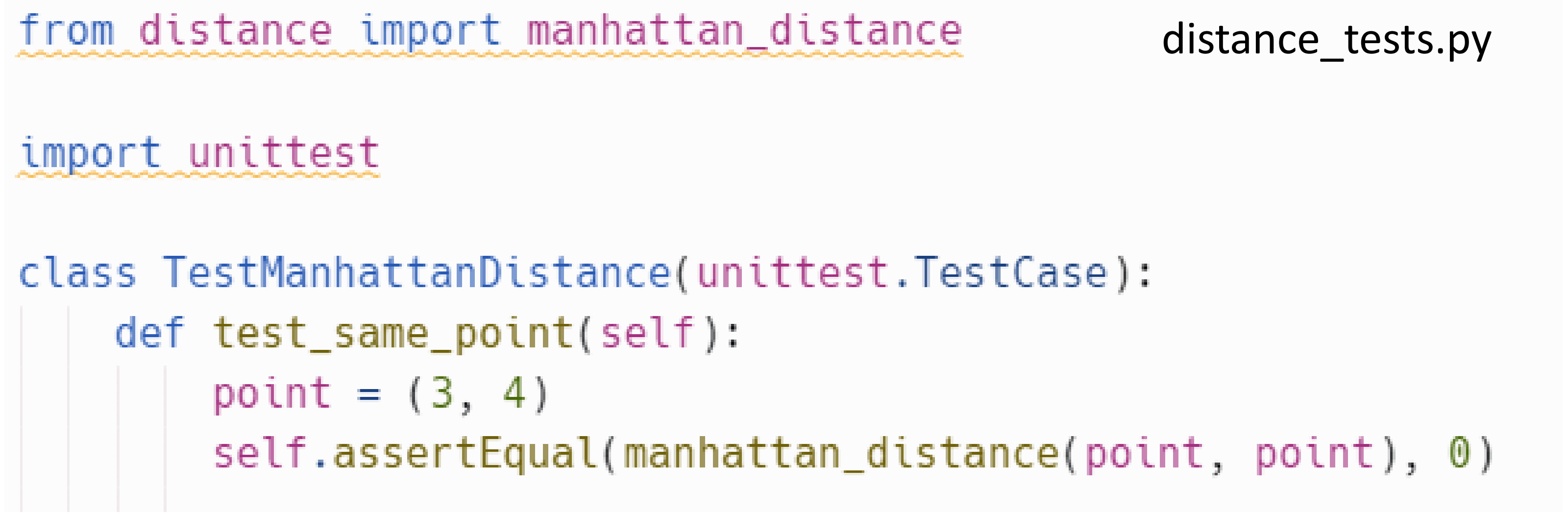
```python
def manhattan_distance(point1, point2):

    x1, y1 = point1
    x2, y2 = point2
    return abs(x1 - x2) + abs(y1 - y2)
```

distance.py



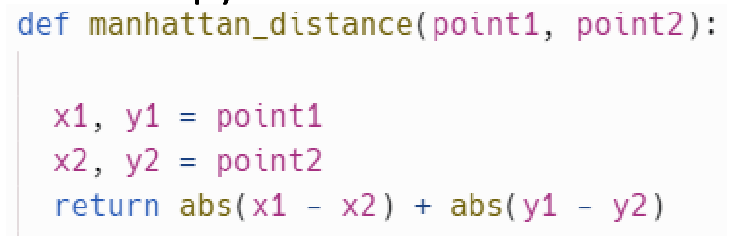
```python
def manhattan_distance(point1, point2):

  x1, y1 = point1
  x2, y2 = point2
  return abs(x1 - x2) + abs(y1 - y2)
```

distance_tests.py



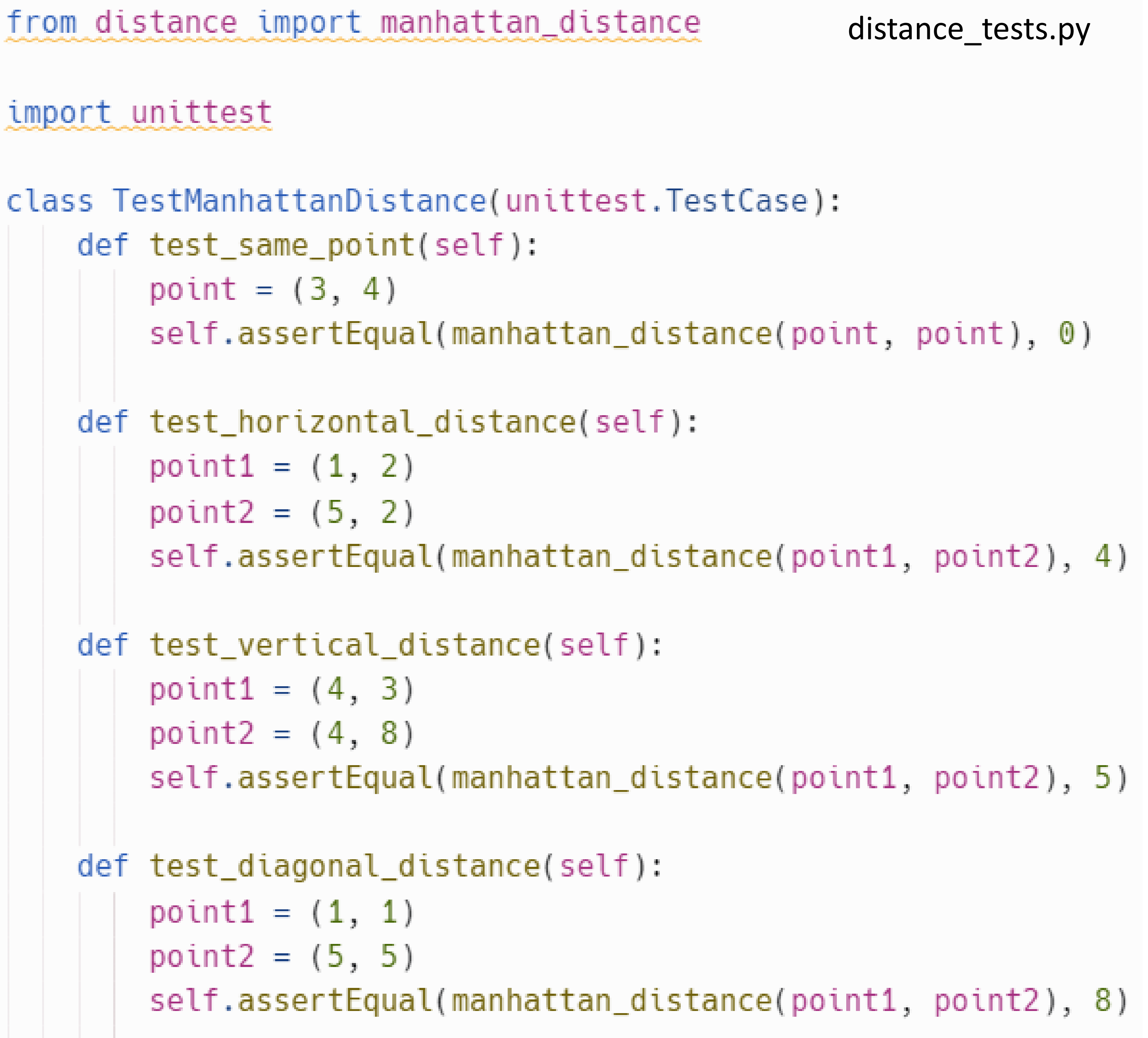
```python
from distance import manhattan_distance

import unittest

class TestManhattanDistance(unittest.TestCase):
    def test_same_point(self):
        point = (3, 4)
        self.assertEqual(manhattan_distance(point, point), 0)
```

distance.py



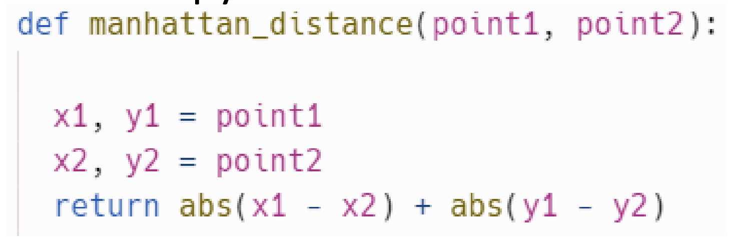
```python
def manhattan_distance(point1, point2):

    x1, y1 = point1
    x2, y2 = point2
    return abs(x1 - x2) + abs(y1 - y2)
```

distance_tests.py



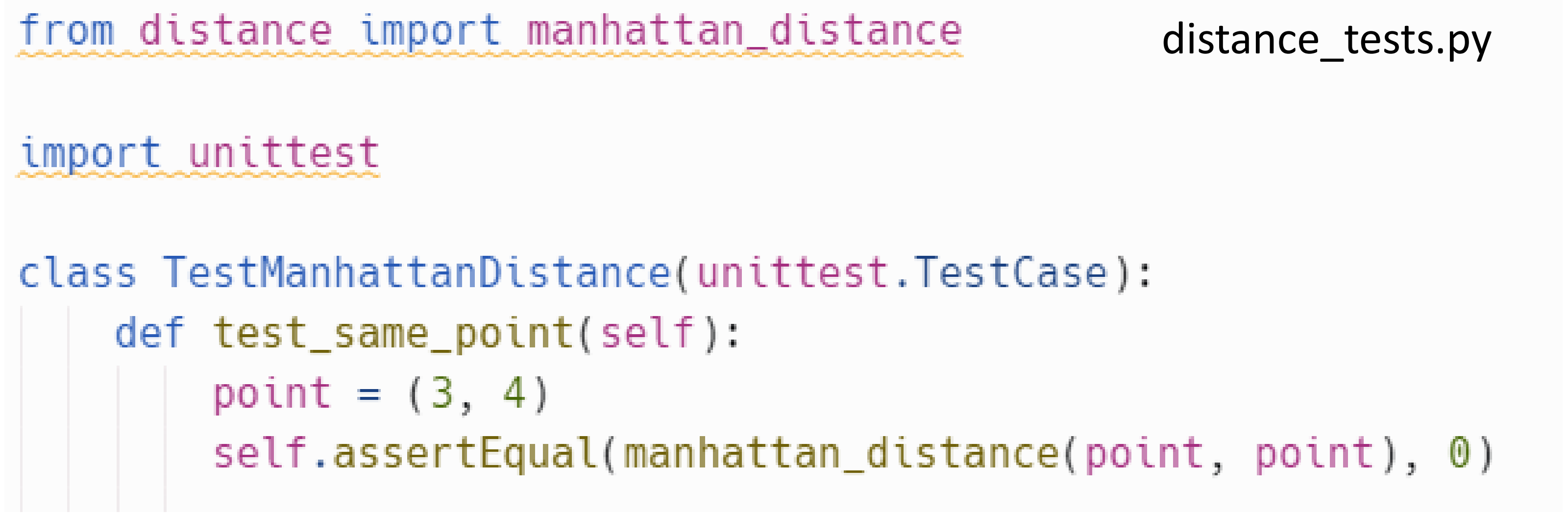
```python
from distance import manhattan_distance

import unittest

class TestManhattanDistance(unittest.TestCase):
    def test_same_point(self):
        point = (3, 4)
        self.assertEqual(manhattan_distance(point, point), 0)

    def test_horizontal_distance(self):
        point1 = (1, 2)
        point2 = (5, 2)
        self.assertEqual(manhattan_distance(point1, point2), 4)

    def test_vertical_distance(self):
        point1 = (4, 3)
        point2 = (4, 8)
        self.assertEqual(manhattan_distance(point1, point2), 5)

    def test_diagonal_distance(self):
        point1 = (1, 1)
        point2 = (5, 5)
        self.assertEqual(manhattan_distance(point1, point2), 8)
```

distance.py



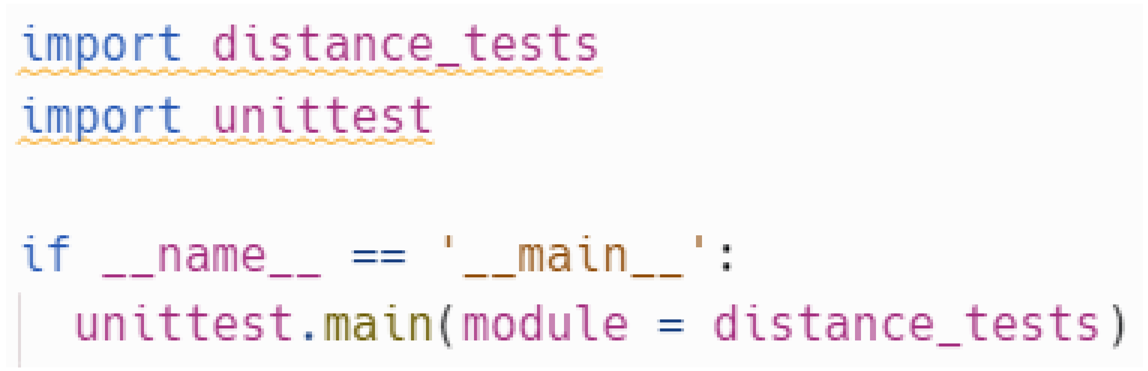
```python
def manhattan_distance(point1, point2):

  x1, y1 = point1
  x2, y2 = point2
  return abs(x1 - x2) + abs(y1 - y2)
```

distance_tests.py



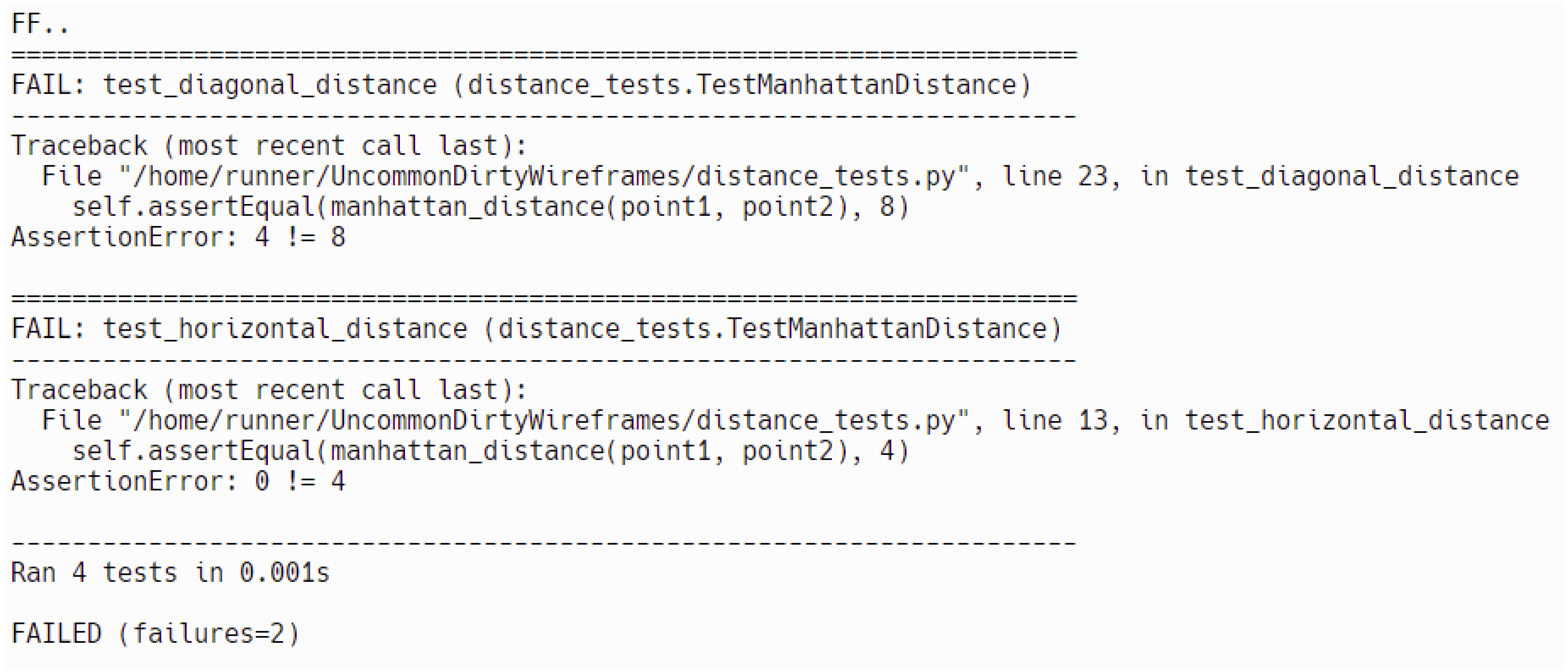
```python
from distance import manhattan_distance

import unittest

class TestManhattanDistance(unittest.TestCase):
    def test_same_point(self):
        point = (3, 4)
        self.assertEqual(manhattan_distance(point, point), 0)
```

main.py

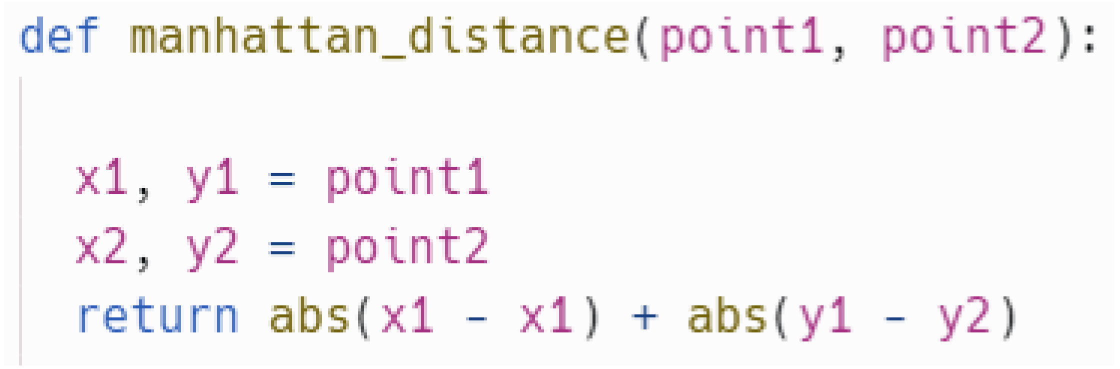


```python
import distance_tests
import unittest

if __name__ == '__main__':
  unittest.main(module = distance_tests)
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.003s

OK
```

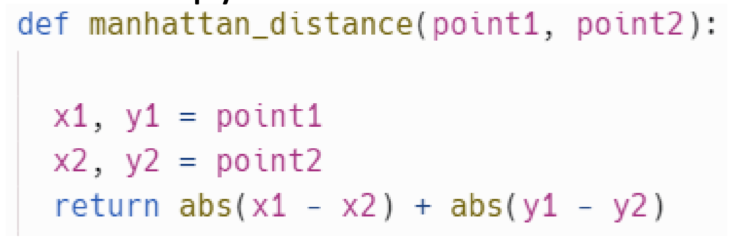


```
FF..
======================================================================
FAIL: test_diagonal_distance (distance_tests.TestManhattanDistance)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/runner/UncommonDirtyWireframes/distance_tests.py", line 23, in test_diagonal_distance
    self.assertEqual(manhattan_distance(point1, point2), 8)
AssertionError: 4 != 8

======================================================================
FAIL: test_horizontal_distance (distance_tests.TestManhattanDistance)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/runner/UncommonDirtyWireframes/distance_tests.py", line 13, in test_horizontal_distance
    self.assertEqual(manhattan_distance(point1, point2), 4)
AssertionError: 0 != 4

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (failures=2)
```



```python
def manhattan_distance(point1, point2):

  x1, y1 = point1
  x2, y2 = point2
  return abs(x1 - x1) + abs(y1 - y2)
```

distance.py

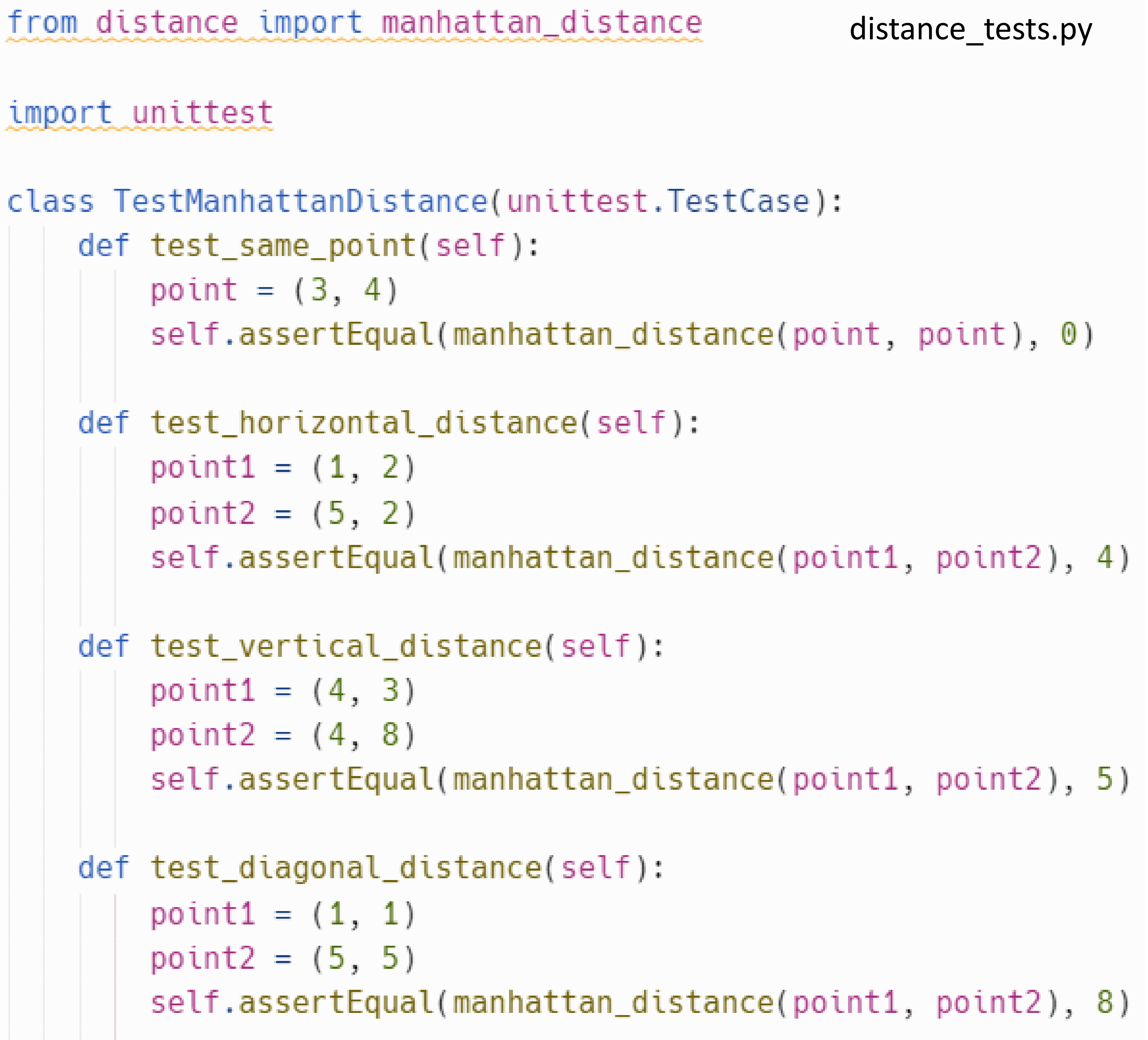


```python
def manhattan_distance(point1, point2):

    x1, y1 = point1
    x2, y2 = point2
    return abs(x1 - x2) + abs(y1 - y2)
```

Think about boundary cases:

E.g. what if there are negative coordinates?

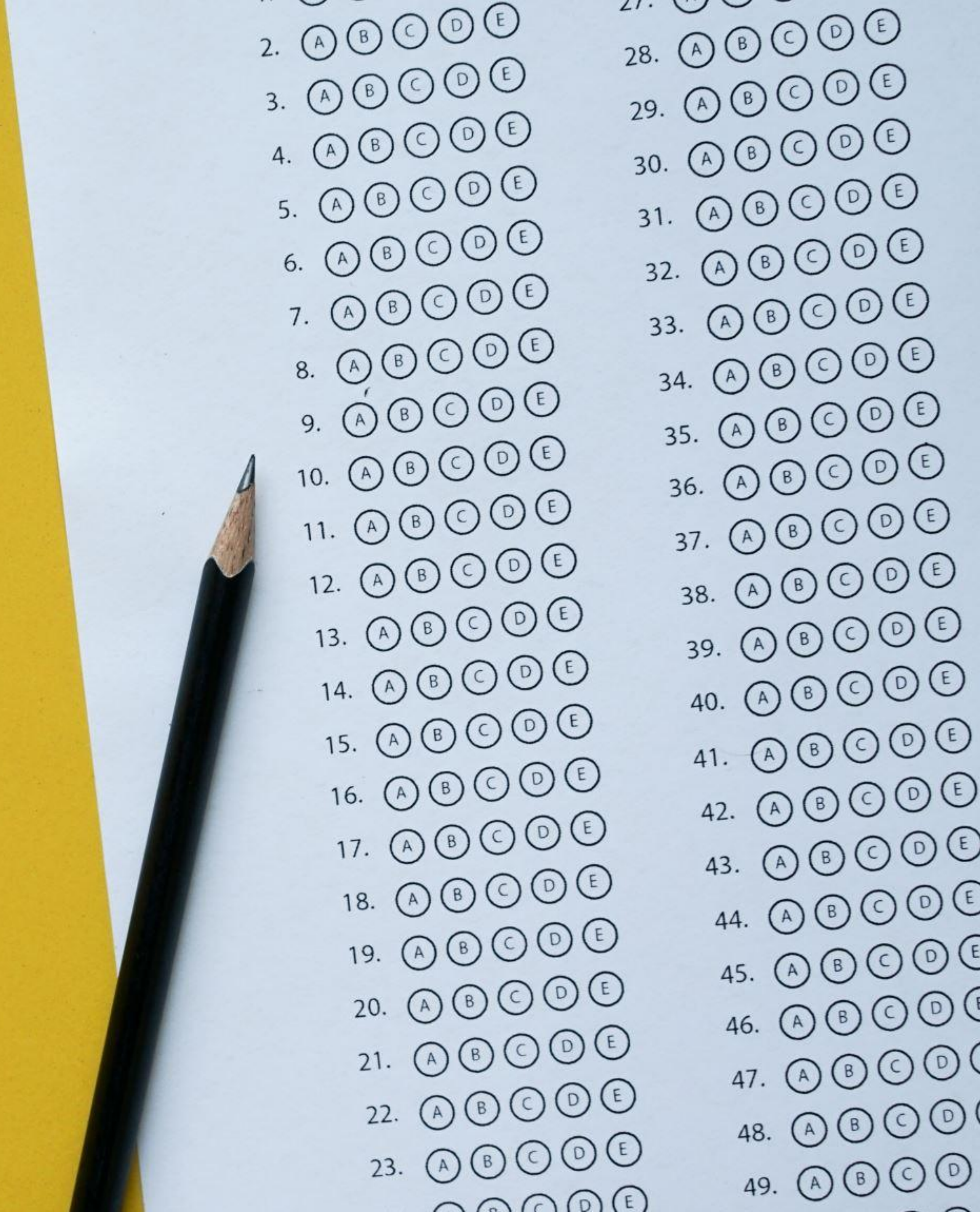
distance_tests.py

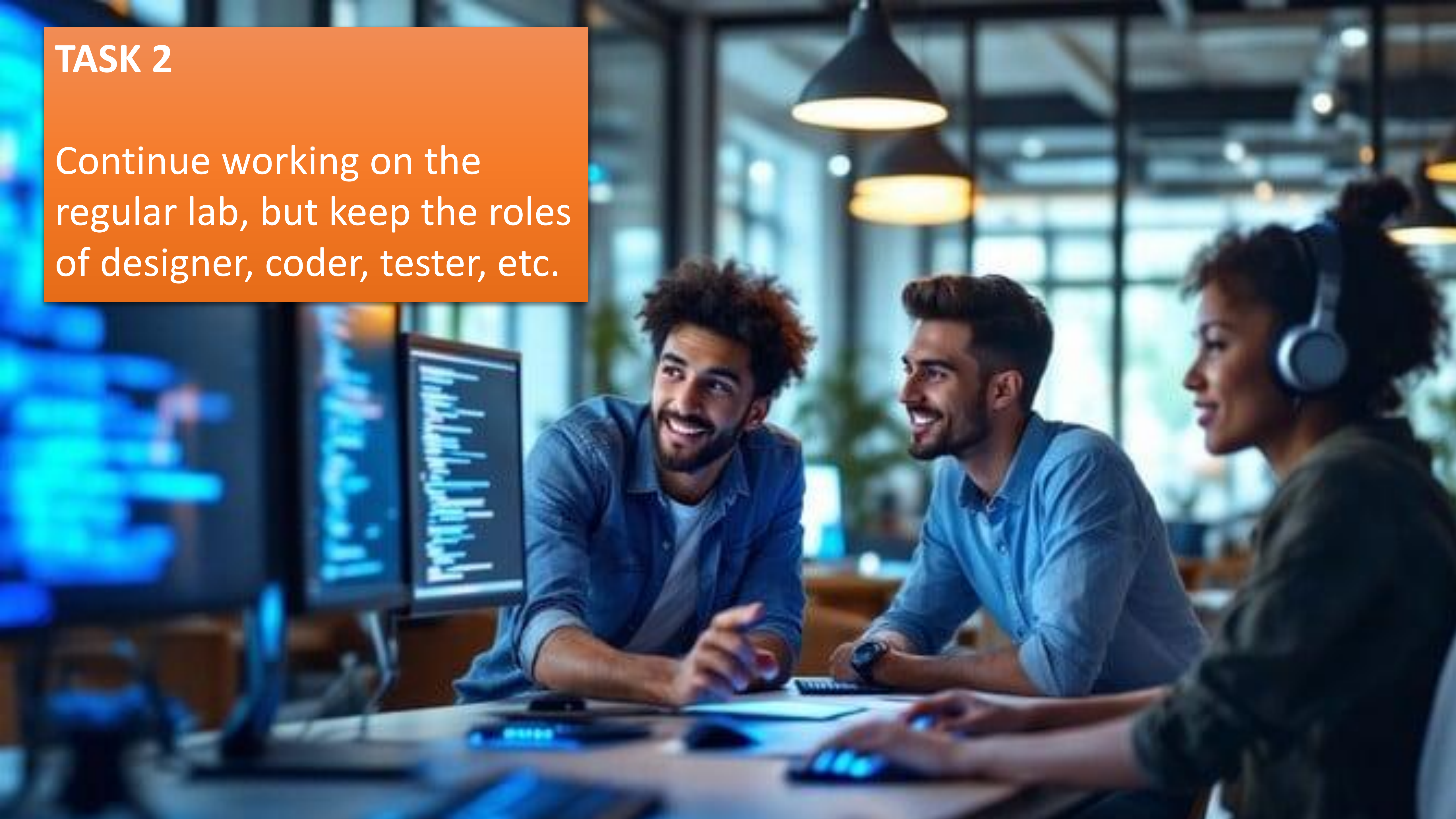


```python
from distance import manhattan_distance

import unittest

class TestManhattanDistance(unittest.TestCase):
    def test_same_point(self):
        point = (3, 4)
        self.assertEqual(manhattan_distance(point, point), 0)

    def test_horizontal_distance(self):
        point1 = (1, 2)
        point2 = (5, 2)
        self.assertEqual(manhattan_distance(point1, point2), 4)

    def test_vertical_distance(self):
        point1 = (4, 3)
        point2 = (4, 8)
        self.assertEqual(manhattan_distance(point1, point2), 5)

    def test_diagonal_distance(self):
        point1 = (1, 1)
        point2 = (5, 5)
        self.assertEqual(manhattan_distance(point1, point2), 8)
```

Test-Driven Development

1. Write a test that defines the desired behavior of a small piece of functionality.
2. Run the test (it should fail because the functionality hasn't been implemented yet).
3. Write the minimum amount of code necessary to pass the test.
4. Run the test again (it should pass now).
5. Refactor the code if necessary while ensuring that all tests still pass.





TASK 2

Continue working on the regular lab, but keep the roles of designer, coder, tester, etc.