CONCEPTS OF PROGRAMMING

MODULE 5
MPCS 50101

THE UNIVERSITY OF CHICAGO

REGULAR EXPRESSION

MODULE 5
MPCS 50101

THE UNIVERSITY OF CHICAGO

REGULAR EXPRESSION

- Regular expression
  - “regex” or “regexp”
  - Concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters
  - Sometimes “too concise” and “too flexible”

REGULAR EXPRESSION

- Many will use regex
- Few will master (or need to)

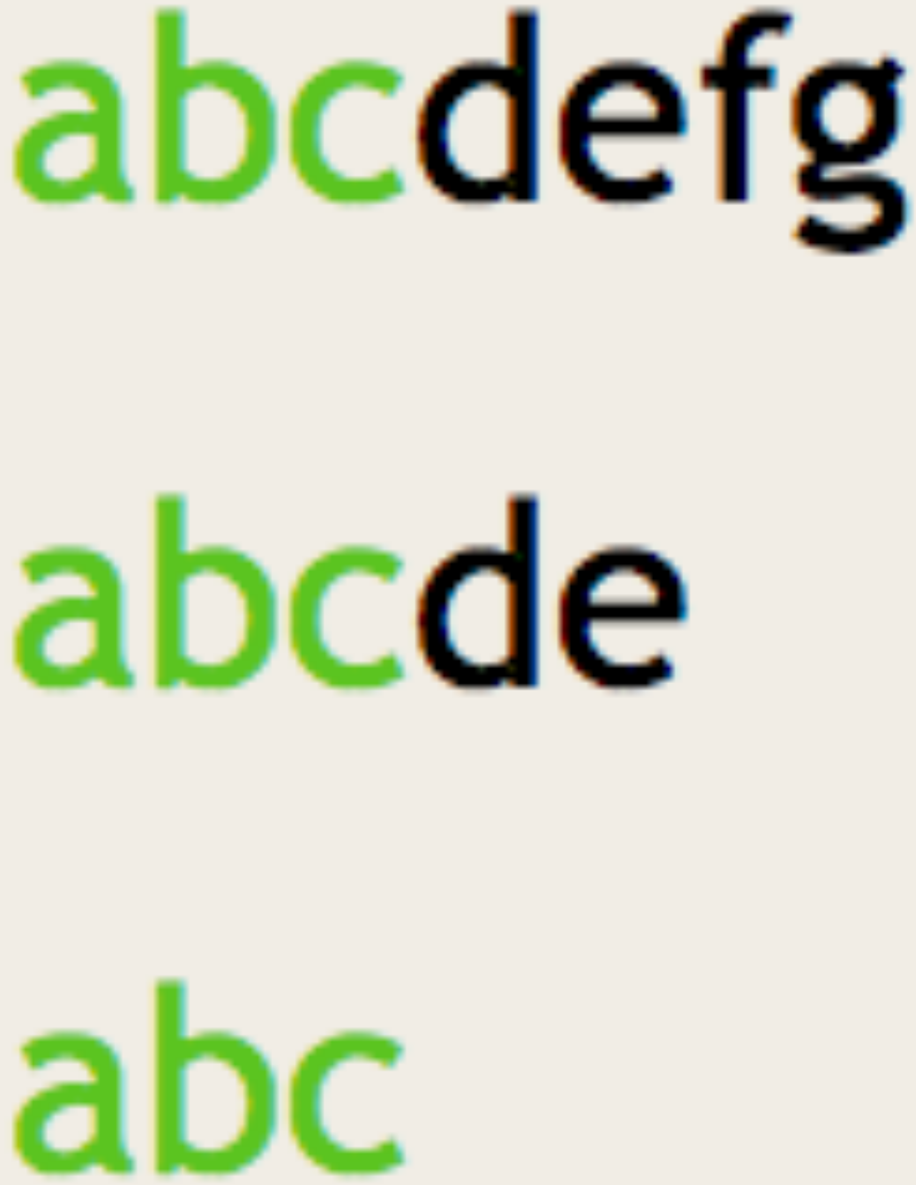
REGULAR EXPRESSION

- A regular expression is written in a formal language that can be interpreted by a regular expression processor
  - Other languages support regular expression
  - Implementation details may differ slightly

REGULAR EXPRESSION

- Match the string "abc"

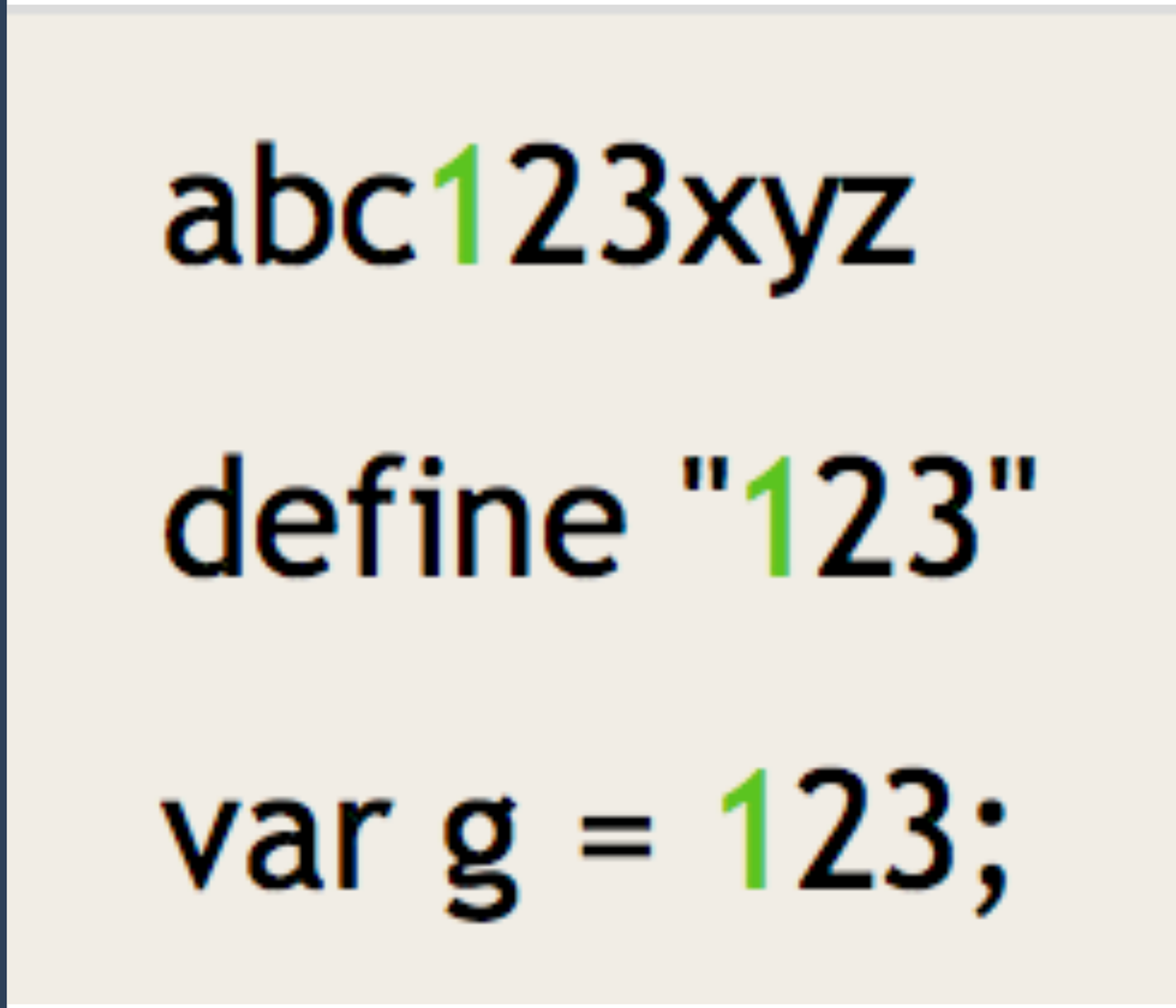
```
abc
```



REGULAR EXPRESSION

- Match the number "1"

1



REGULAR EXPRESSION

- Match the number "123"

123

abc123xyz

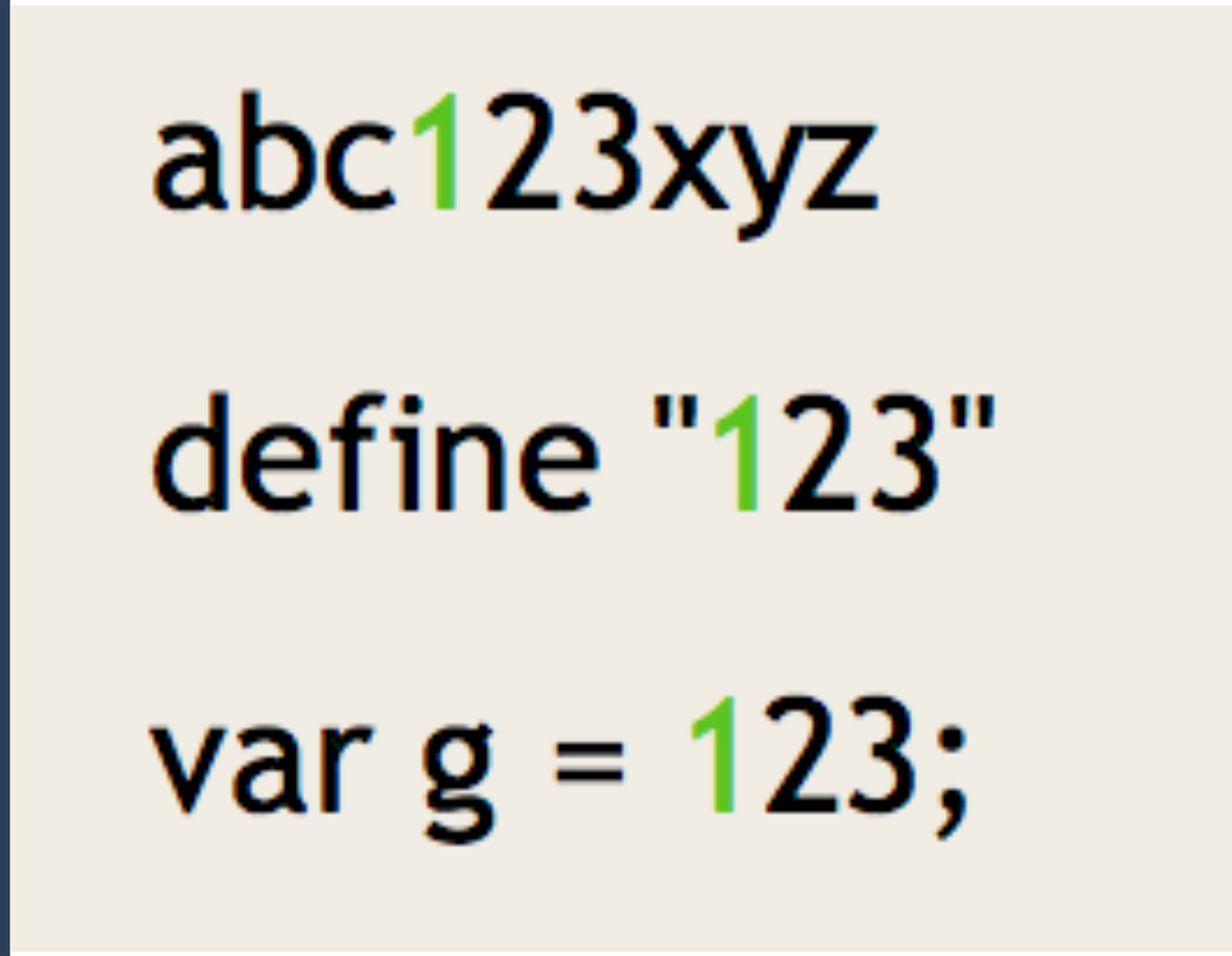
define "123"

var g = 123;

REGULAR EXPRESSION

- Match a single number "\d"
  - Matches first instance

```
\d
```



REGULAR EXPRESSION

- Match 3 numbers "\d{3}"

\d{3}

Match abc123xyz

Match define "123"

Match var g = 123;

REGULAR EXPRESSION

- Match more than 1 members of the following set "[19876542]+"

[19876542]+

abc123xyz

define "123"

var g = 123;

REGULAR EXPRESSION

^	Matches the beginning of a line
$	Matches the end of the line
.	Matches any character
\s	Matches whitespace
\S	Matches any non-whitespace character
*	Repeats a character zero or more times
*?	Repeats a character zero or more times (non-greedy)
+	Repeats a character one or more times
+?	Repeats a character one or more times (non-greedy)
[aeiou]	Matches a single character in the listed set
[^XYZ]	Matches a single character not in the listed set
[a-z0-9]	The set of characters can include a range
(	Indicates where string extraction is to start
)	Indicates where string extraction is to end

REGULAR EXPRESSION

PARSE CALENDAR DATA

```
^(?:(?:(?:0?[13578]|1[02])(\/|-|\.)31)\1|(?:(?:0?[13-9]|
1[0-2])(\/|-|\.)(?:29|30)\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})
$|^(?:0?2(\/|-|\.)29\3(?:(?:(?:1[6-9]|[2-9]\d)?(?:0[48]|
[2468][048]|[13579][26])|(?:(?:16|[2468][048]|[3579]
[26])00))))$|^(?:(?:0?[1-9])|(?:1[0-2]))(\/|-|\.)(?:0?
[1-9]|1\d|2[0-8])\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
```

REGULAR EXPRESSION

RFC822 EMAIL ADDRESS

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
)+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:
\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(
?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\0
31]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\
](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+
(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:
(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)
?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\
r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
 \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)
?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t]
)*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
 \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*
)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
```

REGULAR EXPRESSION

- Before you can use regular expressions in your program, you must import the library using **`import re`**

```
#
# Secret Labs' Regular Expression Engine
#
# re-compatible interface for the sre matching engine
#
# Copyright (c) 1998-2001 by Secret Labs AB.  All rights reserved.
#
# This version of the SRE library can be redistributed under CNRI's
# Python 1.6 license.  For any other use, please contact Secret Labs
# AB (info@pythonware.com).
#
# Portions of this engine have been developed in cooperation with
# CNRI.  Hewlett-Packard provided funding for 1.6 integration and
# other compatibility work.
#

r"""Support for regular expressions (RE).

This module provides regular expression matching operations similar to
those found in Perl.  It supports both 8-bit and Unicode strings; both
the pattern and the strings being processed can contain null bytes and
characters outside the US ASCII range.

Regular expressions can contain both special and ordinary characters.
Most ordinary characters, like "A", "a", or "0", are the simplest
regular expressions; they simply match themselves.  You can
concatenate ordinary characters, so last matches the string 'last'.

The special characters are:
```

REGULAR EXPRESSION

```
import re

saying = "One fish, two fish, red fish blue fish.  This one has a
little star, this one has a little car.  Say, what alot of fish
there are."


if re.search('fish', saying):
    print("Found the fish")

# Return an array
print((re.findall('fish',saying)))
```

REGULAR EXPRESSION

- You can use **re.search()** to see if a string matches a regular expression
  - Similar to find() in strings

```
if re.search('fish', saying):
    print("Found the fish")
```

```
match = re.search("abc","abcdefghi")
print(match)
```

```
>> <re.Match object; span=(0, 3),
match='abc'>
```

REGULAR EXPRESSION

- **`re.findall()`** to extract portions of a string that match your regular expression

```
import re

saying = "One fish, two fish, red fish
blue fish.  This one has a little star,
this one has a little car.  Say, what
alot of fish there are."

if saying.find('wish'):
    print "Found it"

if re.search('fish', saying):
    print "Found the fish"

# Return all matches
print re.findall('fish',saying)
# >> ['fish', 'fish', 'fish', 'fish',
'fish']
```

REGULAR EXPRESSION

```
import re
saying = "One fish, two fish, red fish blue fish.  This one
has a little star, this one has a little car.  Say, what alot
of fish there are."

# Multiple match
match = re.findall('fish',saying)
if match:
    print(("Found:", match))
else:
    print("Didn't find it")
```

REGULAR EXPRESSION

```
import re

#In raw string literals, backslashes have no special meaning as an escape
character. This is useful for writing strings that contain a lot of
backslashes. Not having to escape each backslash makes the string more
readable.

str = r'\User\timothy\files'
match = re.search(r'\\timothy', str)

# If-statement after search() tests if it succeeded
# results are in the .group()
if match:
    print('found', match.group())
else:
    print('did not find')
```

REGULAR EXPRESSION

```
## Search for pattern 'iii' in string 'piiig'.
## All of the pattern must match, but it may appear
## anywhere.
## On success, match.group() is matched text.

match = re.search(r'iii', 'piiig')
#  found, match.group() == "iii"

match = re.search(r'igs', 'piiig')
#  not found, match == None
```

REGULAR EXPRESSION

```
## . = any char but \n
match = re.search(r'..g', 'piiig')

# =>  found, match.group() == “iig"
```

REGULAR EXPRESSION

```
## Matching digits or characters
## \d = digit char,
## \w = word char

match = re.search(r'\d\d\d', 'p123g') =>  found,
match.group() == "123"

match = re.search(r'\w\w\w', '@@abcd!!') =>  found,
match.group() == "abc"
```

REGULAR EXPRESSION

```
match = re.search(r'\d\d\d', 'p123g')
print match.group() # "123"

# Greedy match
match = re.search(r'\d+', 'p123g')
print match.group() # "123"

match = re.search(r'\d{5}', 'p123456789g')
print match.group() # "12345"

match = re.search(r'\w\w\w', '@@abcd!!')
print match.group() # 'abc'

match = re.search(r'\w{4}', '@@abcd!!')
print match.group() # 'abcd'

match = re.search(r'(\d\w){3}', '1a2a3a4a5a')
print match.group() # 1a2a3a
```

REGULAR EXPRESSION

GROUP CAN CONTAIN MORE THAN A SINGLE RESULT

GROUPS ()

```
import re

regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")

print(("Match at index %s, %s" % (match.start(), match.end())))

# So this will print("June 24"
print(("Full match: %s" % (match.group(0))))

# So this will print("June"
print(("Month: %s" % (match.group(1))))

# So this will print("24"
print(("Day: %s" % (match.group(2))))
```

GROUP 1 MATCH

GROUP 2 MATCH

REGULAR EXPRESSION

RegexOne Interactive Tutorial References & More

Lesson 1½: The 123s

Characters include normal letters, but digits as well. In fact, numbers 0-9 are also just characters and if you look at an ASCII table, they are listed sequentially.

Over the various lessons, you will be introduced to a number of special metacharacters used in regular expressions that can be used to match a specific type of character. In this case, the character **\d** can be used in place of **any digit from 0 to 9**. The preceding slash distinguishes it from the simple **d** character and indicates that it is a metacharacter.

HTTPS://REGEXONE.COM/LESSON/INTRODUCTION_ABCS

Below are a few more lines of text containing digits. Try writing a pattern that matches all

UNIT TESTING

MODULE 5
MPCS 50101

THE UNIVERSITY OF CHICAGO

UNIT TESTING

- Method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use

UNIT TESTING

- Code that runs to check the validity of other code

UNIT TESTING

REQUIRES SOME UPFRONT WORK, BUT PAYS OFF IN THE END

- Unit testing is a method to help you write more robust code
  - Proves that code works the way you think it should
  - Protect against future changes
  - Makes you think about what can go wrong
  - Simplify the write, try, edit, fix, write, try, edit, fix, cycle
  - Documents how the code is used

UNIT TESTING

- Proves that code works the way you think it should

```
def sum(x,y):
    total = x + y
    return total


sum(1,3) == 4 👍

sum(1,2) != 4 👎
```

UNIT TESTING

- Makes you think about what can go wrong

```
def sum(x,y):
    total = x + y
    return total


sum("hello","world") == "hello world"👍

sum("hello", 3) 👎
```

UNIT TESTING

- Simplify the write, try, edit, fix, write, try, edit, fix, cycle
- Documents how the code is used

```
def sum(x,y):
    total = x + y
    return total


sum(1,3) == 4 👍
sum(10,10) == 20 👍
sum("hello","world") == "hello world"👍
```

UNIT TESTING

- Protect against future changes

```
def sum(x,y):
    x = int(x)
    y = int(y)
    total = x + y
    return total

sum(1,3) == 4 👍
sum(10,10) == 20 👍
sum("hello","world") == "hello world"👎
```

UNIT TESTING

- Python testing frameworks automate testing code
- Separate programs and tools that run alongside your code

Unit Testing Tools

Tool	Category	Author	Part of	Claim to f
unittest	unit testing	Steve Purcell	Python standard lib	first unit te people far reuse via
doctest	unit testing	Tim Peters	Python standard lib	copy and themselve Django's a
pytest	unit testing	Holger Krekel	It used to be named py.test which was part of the pylib. Standalone now.	no API!;au fixture/stat customize
nose	unittest extensions	Jason Pellerin		unit test fr an alterna mimic the too much are also m
testify	unittest extensions	Yelp team		unit test fr buckets fo with lots o

UNIT TESTING

- Most popular
  - Unittest
  - Nose
  - pytest

PART OF PYTHON USED TO TEST THE LANGUAGE

Unit Testing Tools

Tool	Category	Author	Part of	Claim to f
unittest	unit testing	Steve Purcell	Python standard lib	first unit te people far reuse via
doctest	unit testing	Tim Peters	Python standard lib	copy and themselve Django's a
t	unit testing	Holger Krekel	It used to be named py.test which was part of the pylib. Standalone now.	no API!;au fixture/sta customize
	unittest extensions	Jason Pellerin		unit test fr an alterna mimic the too much are also m
testify	unittest extensions	Yelp team		unit test fr buckets fo with lots o
				Extension

PYTEST

PYTEST

- Easy to use
- Scalable
- Detailed info on failing assert statements
- Auto-discovery of test modules and functions
- Support Python 2 and 3
- Rich plugin architecture

PYTEST

```
1 == 1
```

True

PYTEST

```
10 != 2
```

True

PYTEST

```
10 > 100
```

False

PYTEST

```
10 >= 10
```

True

PYTEST

```
"cat" in "catalog"
```

True

PYTEST

```
1 == 1                          True
10 != 2                         True
10 > 100                        False
10 >= 10                        True
"cat" in "catalog"              True
```

PYTEST

```
assert 1 == 1                          True
assert 10 != 2                         True
assert 10 > 100                        False
assert 10 >= 10                        True
assert "cat" in "catalog"              True
```

PYTEST

```
# Should be installed as part of anaconda, but just in
# case

pip install pytest
```

PYTEST

```
543 % pytest
============================================= test session starts
=============================================
platform darwin -- Python 3.7.5, pytest-5.3.2, py-1.8.1, pluggy-0.13.1
rootdir: /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchicago.codes-
courses/mpcs50101/mpcs50101-2020-winter/mpcs501010-2020-winter-coursework/assignment-3
plugins: astropy-header-0.1.1, arraydiff-0.3, remotedata-0.3.2, openfiles-0.4.0,
doctestplus-0.5.0, hypothesis-4.54.2
collected 0 items
```

PYTEST

calculator.py

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

test_calculator.py

```
import calculator

def test_add():
    assert calculator.add(1,3) == 4
```

MAKE ASSERTIONS THAT WE KNOW ARE TRUE

PYTEST


```
2  import pytest
3
   ✓ Run Test | ✓ Debug Test
4  def test_add():
5      assert calculator.add(1,3) == 4
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 3: Python

```
(base) 1-simple-testing$ pytest
============================================================ test session starts ============================================================
platform darwin -- Python 3.7.5, pytest-5.3.2, py-1.8.1, pluggy-0.13.1
rootdir: /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchicago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-20
20-winter-sessions/2020-session-5/1-simple-testing
plugins: astropy-header-0.1.1, arraydiff-0.3, remotedata-0.3.2, openfiles-0.4.0, doctestplus-0.5.0, hypothesis-4.54.2
collected 3 items

test_calculator.py ...                                                                                                          [100%]

============================================================ 3 passed in 0.02s =============================================================
(base) 1-simple-testing$
```

PYTEST

PYTEST -K TEST_NAME

```
2  import pytest
3
   ✓ Run Test | ✓ Debug Test
4  def test_add():
5      assert calculator.add(1,3) == 4
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 3: Python

```
(base) 1-simple-testing$ pytest -k test_add
============================= test session starts ==============================
platform darwin -- Python 3.7.5, pytest-5.3.2, py-1.8.1, pluggy-0.13.1
rootdir: /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchicago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-2020-winter-sessions/2020-session-5/1-simple-testing
plugins: astropy-header-0.1.1, arraydiff-0.3, remotedata-0.3.2, openfiles-0.4.0, doctestplus-0.5.0, hypothesis-4.54.2
collected 3 items / 2 deselected / 1 selected

test_calculator.py .                                                     [100%]

======================= 1 passed, 2 deselected in 0.02s ========================
(base) 1-simple-testing$
```

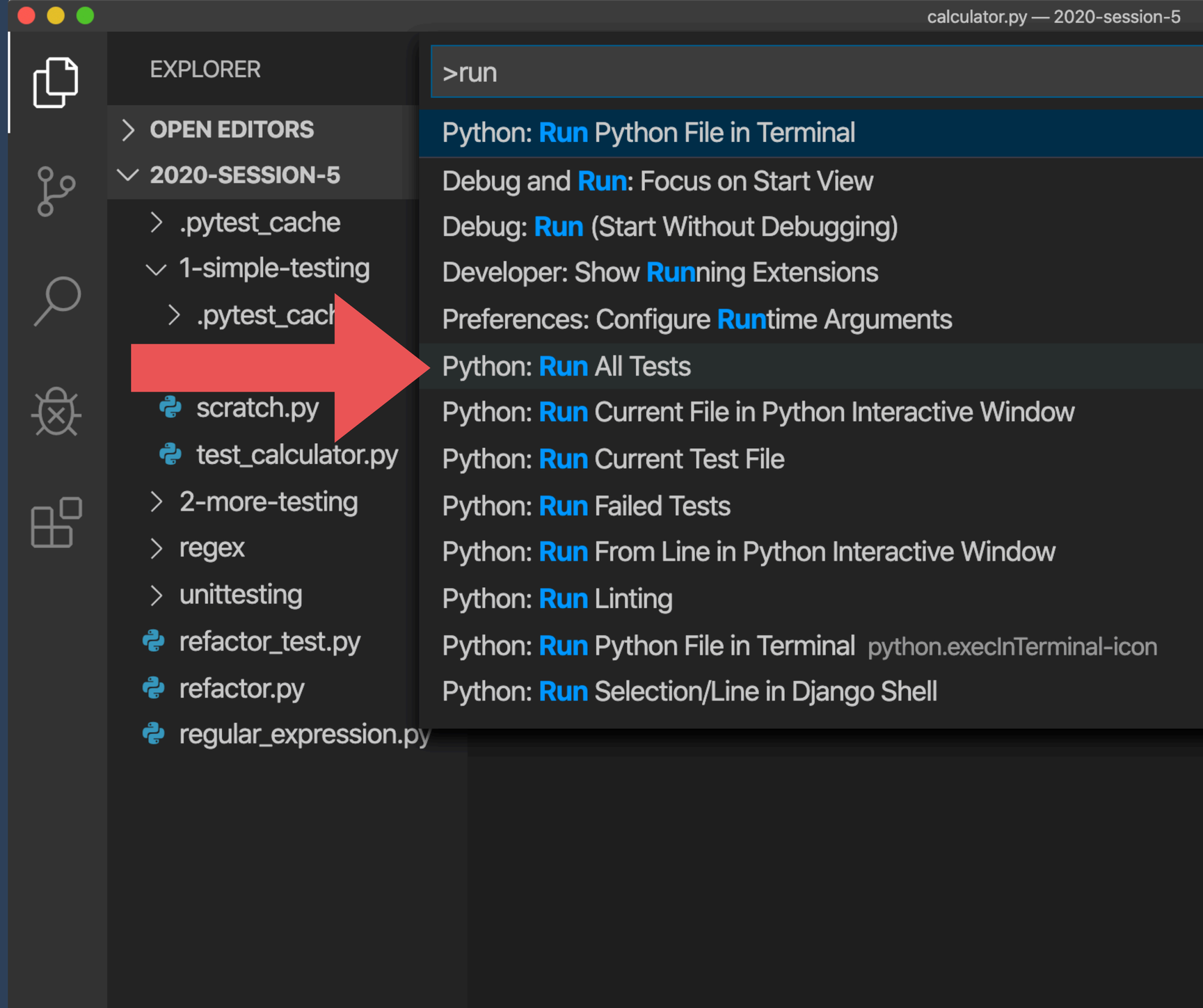
PYTEST

```
# pytest runs on files following naming conventions

pytest test_*.py

pytest tests/

pytest -k test_name

# many more options ...
```

TESTING IN VSCODE

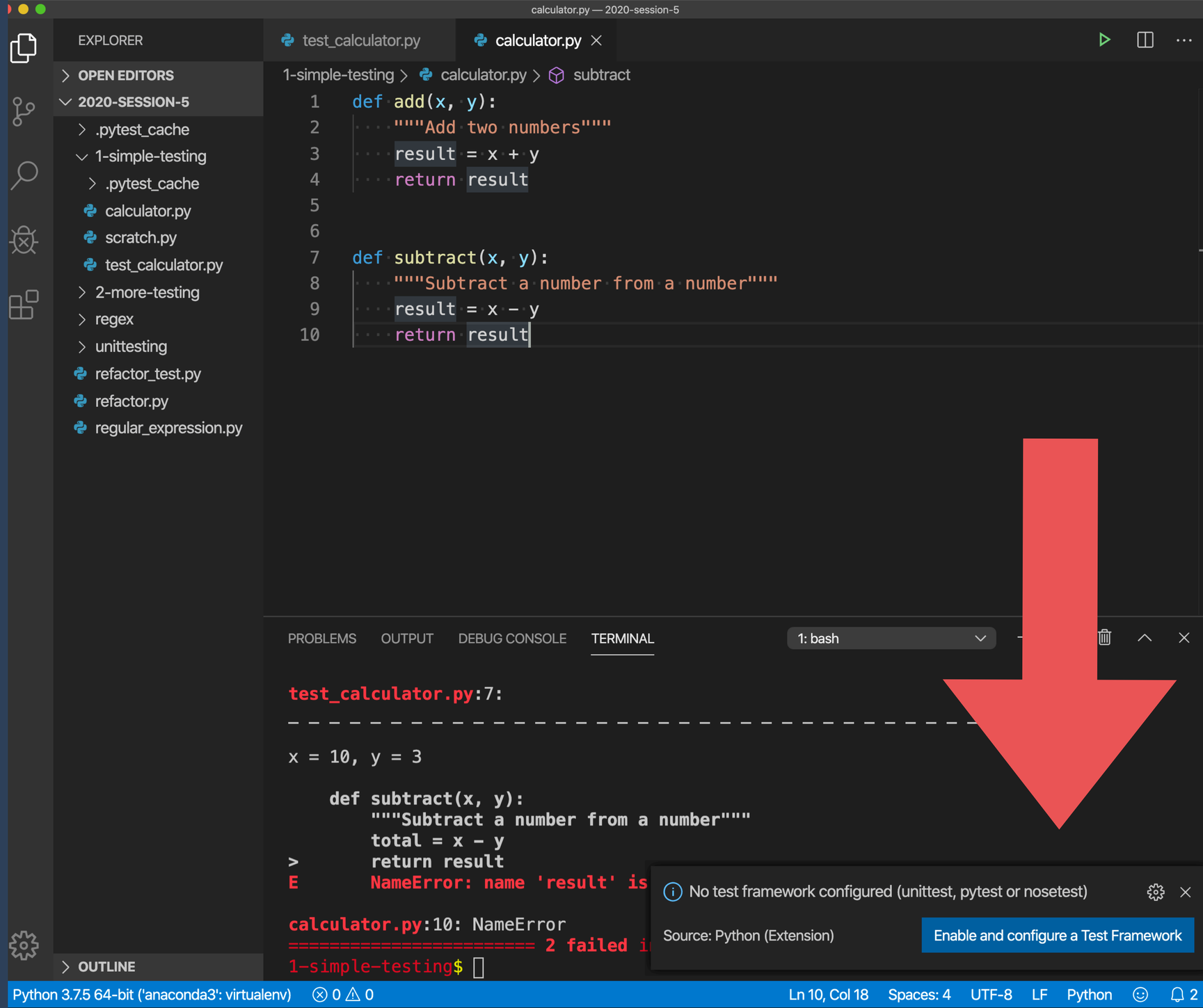
TESTING IN VSCODE

- Enable testing in VSCode
- CMD-Shift-P > Run All Test

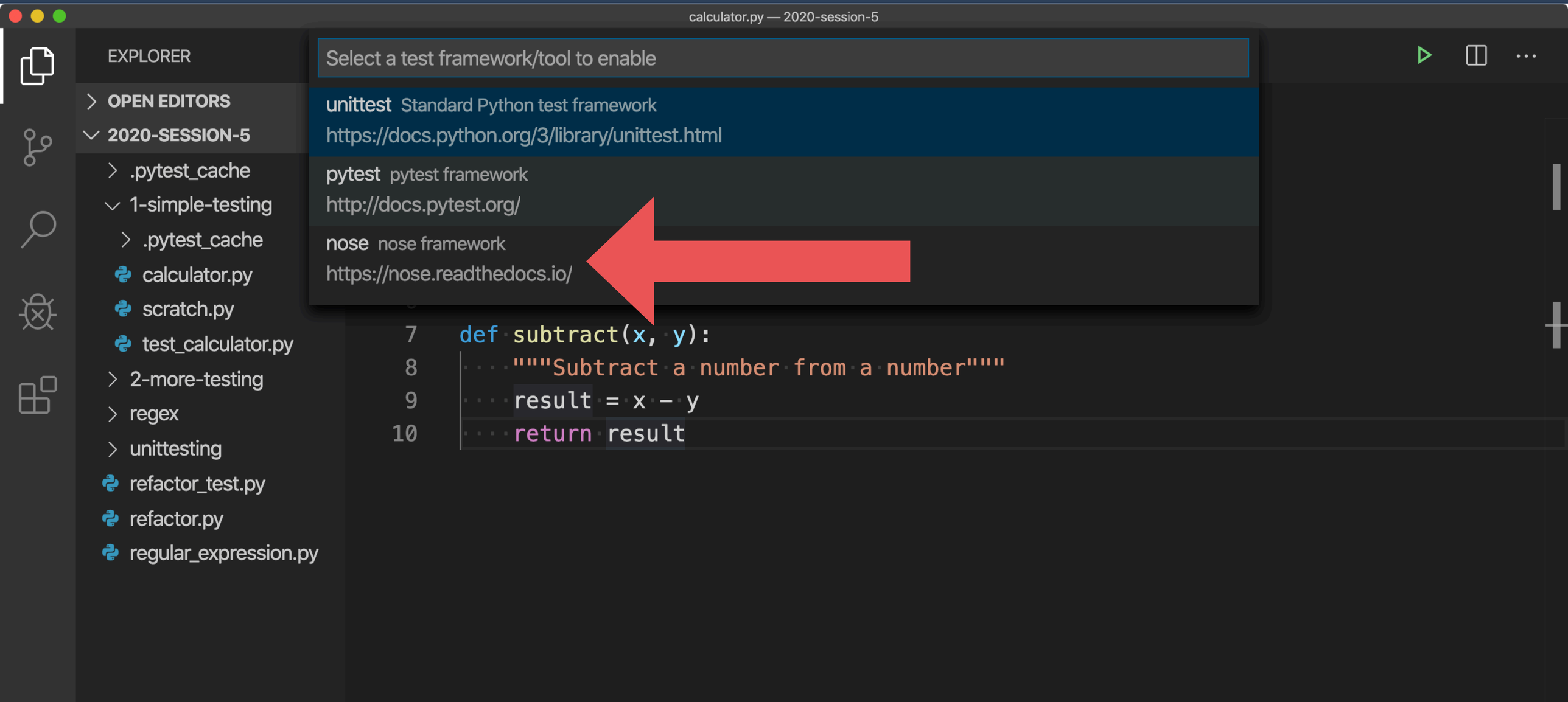


TESTING IN VSCODE

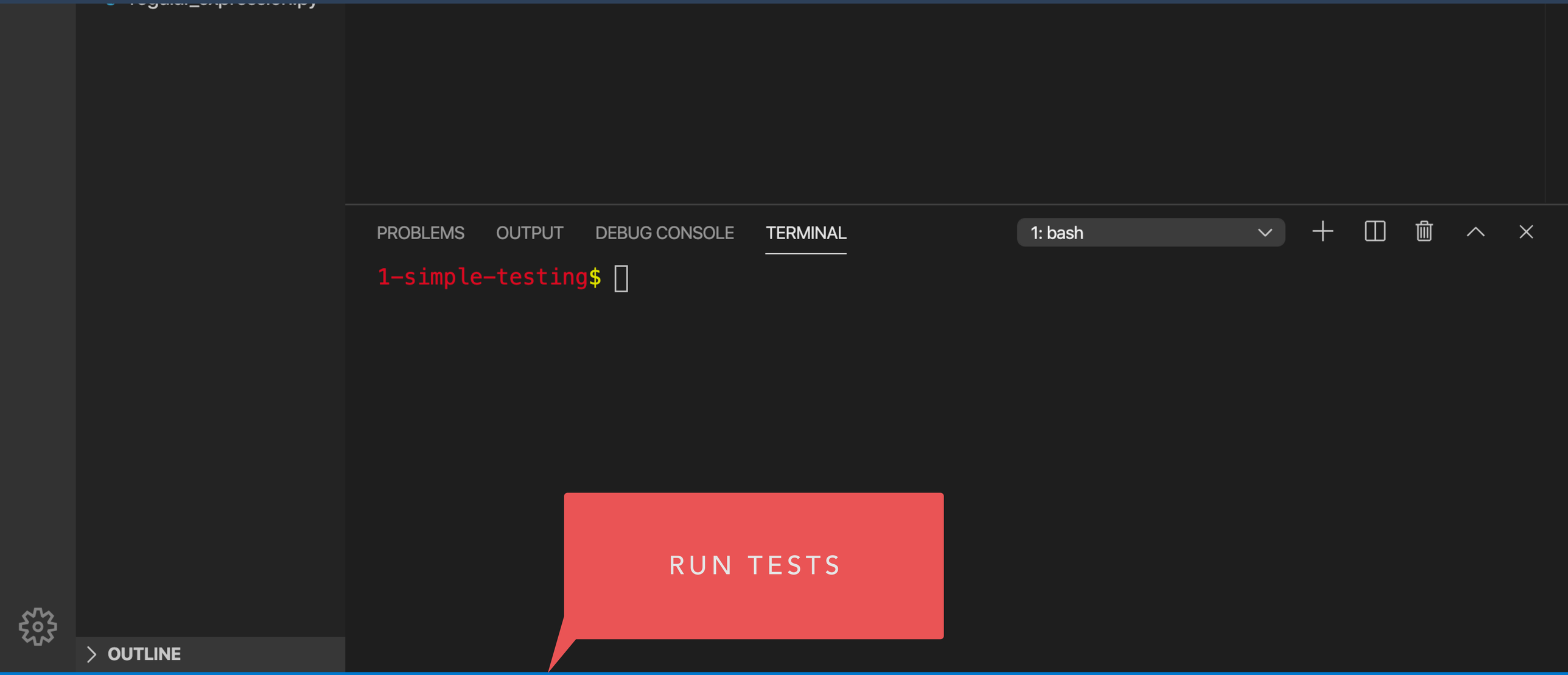
- Enable testing framework



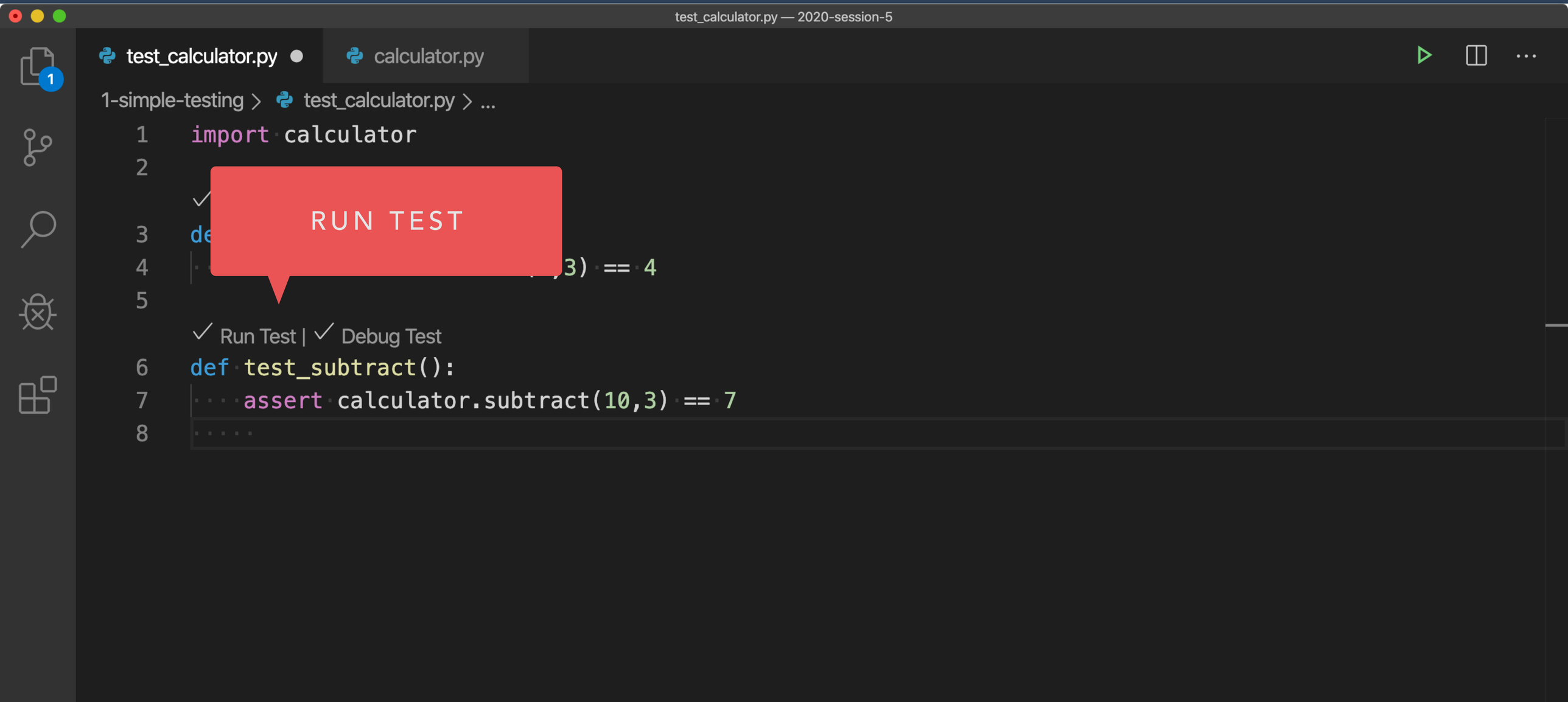
TESTING IN VSCODE



TESTING IN VSCODE

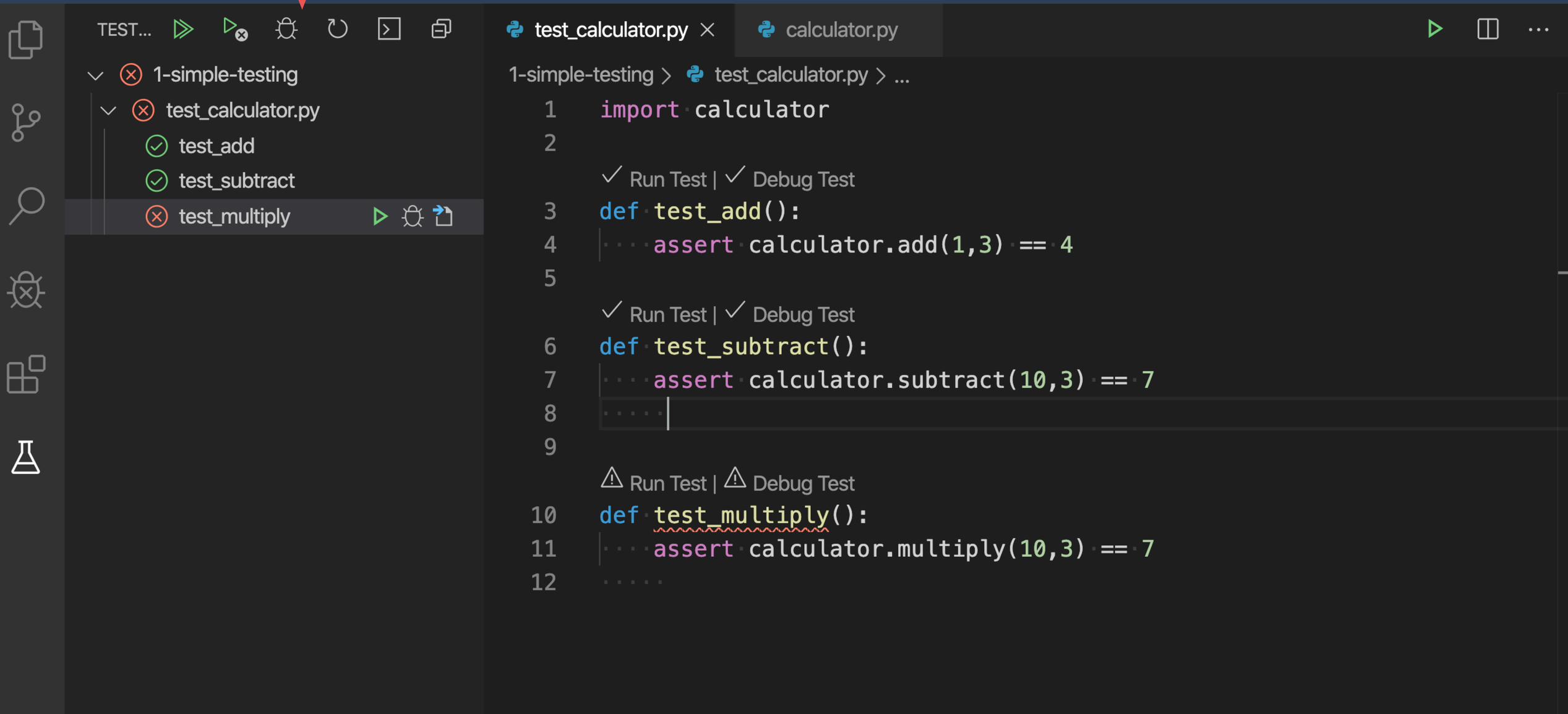


TESTING IN VSCODE



TESTIN DE

TESTING PANE



CALCULATOR TEST

CALCULATOR TEST

- Calculator module
  - Add
  - Subtract
  - Divide
  - multiply

```
def add(x, y):
    """Add two numbers"""
    result = x + y
    return result

def subtract(x, y):
    """Subtract a number from a number"""
    result = x - y
    return result

def multiply(x, y):
    """Multiply two numbers"""
    result = x * y
    return result

def divide(x, y):
    """Divide a number by a number"""
    result = x / y
    return result
```

CALCULATOR TEST

```python
import calculator
import pytest

def test_add():
    assert calculator.add(1,3) == 4
    assert calculator.add(10,10) == 20
    assert calculator.add(10,100) != 20
```

TEST THE NEGATIVE CASE

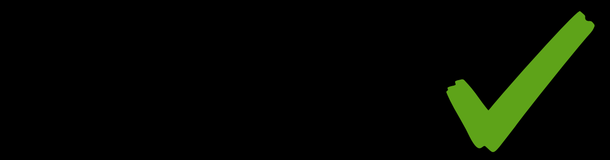
CALCULATOR TEST

```python
import calculator
import pytest

def test_add():
    assert calculator.add(1,3) == 4
    assert calculator.add(10,10) == 20
    assert calculator.add(10,100) == 20
```

FAIL

```
==================== 1 failed, 3 passed, 1 warning in 0.04s ====================
```

CALCULATOR TEST



```python
import calculator
import pytest

def test_multiply():
    assert calculator.multiply(10,3) == 30
    assert calculator.multiply(10,3) != 7
    assert calculator.multiply(0,3) == 0
```

CALCULATOR TEST

```
import calculator
import pytest


def test_divide():
    assert calculator.divide(9,3) == 3

    assert calculator.divide(9,0) == ???
```

IF THIS HAPPENS WE GET AN ERROR, NOT AN INCORRECT VALUE

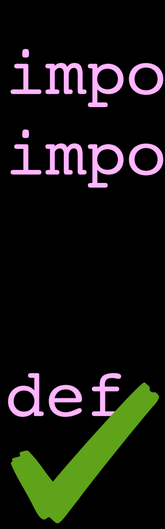
CALCULATOR TEST

```
import calculator
import pytest


def test_divide():
    assert calculator.divide(9,3) == 3


    with pytest.raises(ZeroDivisionError):
        calculator.divide(10, 0)
```

USING **PYTEST.RAISES** IN A WITH BLOCK AS A CONTEXT MANAGER, WE CAN CHECK THAT AN EXCEPTION IS "CORRECTLY" RAISED

CALCULATOR TEST

```
import calculator
import pytest


def test_divide():
    assert calculator.divide(9,3) == 3


    with pytest.raises(ZeroDivisionError):
        calculator.divide(10, 1)
```



THIS TEST WILL **FAIL** BECAUSE IT DID NOT RAISE AN EXCEPTION

CALCULATOR TEST

SAVES THE EXCEPTION OBJECT SO YOU CAN EXTRACT DETAILS FROM IT

```
import calculator
import pytest


def test_divide():
    assert calculator.divide(9,3) == 3
    with pytest.raises(ZeroDivisionError) as excinfo:
        calculator.divide(10, 0)
    assert str(execinfo.value) == 'some info'
```

CALCULATOR TEST

```
def multiply(x, y):
    """Multiply two numbers"""
    result = x * y
    return result
```

```
import calculator
import pytest

def test_multiply():
    assert calculator.multiply(10,3) == 30
    assert calculator.multiply(10,3) != 7
    assert calculator.multiply(0,3) == 0


    with pytest.raises(TypeError):
        calculator.multiply(3, "ten")
```

CALCULATOR TEST

```
def multiply(x, y):
    """Multiply two numbers"""
    result = x * y
    return result
```

```
import calculator
import pytest

def test_multiply():
    assert calculator.multiply(10,3) == 30
    assert calculator.multiply(10,3) != 7
    assert calculator.multiply(0,3) == 0

    assert calculator.multiply(3, "ten") == "tententen"
```

UNEXPECTED RESULT; MAKES YOU THINK

UNITTEST

FOR
REFERENCE

UNIT TESTING

- There is a `unittest` module that provides testing features
- Follow the the boilerplate and update for your needs

```
import unittest
import YOUR_MODULE


class TestSOMETHING(unittest.TestCase):
 """ Boilerplate for unit testing """

 def test_SOMETHING(self):
     self.assertEqual(mod.func(),True)


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

UNIT TESTING

```
#
# simple.py
#
def is_true():
    return True

def is_even(number):
    if number % 2 == 0:
        return True

def divide_by(x,y):
    return x/y
```

UNIT TESTING

NAME MODULE_TEST.PY

```
# simple_test.py

import unittest
import simple

class TestSimple(unittest.TestCase):

    def test_is_true(self):
        self.assertEqual(simple.always_returns_true(),True)

    def test_is_even(self):
        self.assertEqual(simple.is_even(2),True)
        self.assertNotEqual(simple.is_even(3),True)

    def test_divide_by(self):
        """Test exception raise due to run time error"""
        self.assertRaises(ZeroDivisionError, divide_by(5,3))

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

UNIT TESTING

IMPORT UNITTEST AND MODULES YOU ARE TESTING

```
# simple_test.py

import unittest
import simple

class TestSimple(unittest.TestCase):

    def test_is_true(self):
        self.assertEqual(simple.always_returns_true(),True)

    def test_is_even(self):
        self.assertEqual(simple.is_even(2),True)
        self.assertNotEqual(simple.is_even(3),True)

    def test_divide_by(self):
        """Test exception raise due to run time error"""
        self.assertRaises(ZeroDivisionError, divide_by(5,3))

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

UNIT TESTING

BOILERPLATE CLASS DEFINITION
USES TEST PREFIX

```
# simple_test.py

import unittest
import simple

class TestSimple(unittest.TestCase):

    def test_is_true(self):
        self.assertEqual(simple.always_returns_true(),True)

    def test_is_even(self):
        self.assertEqual(simple.is_even(2),True)
        self.assertNotEqual(simple.is_even(3),True)

    def test_divide_by(self):
        """Test exception raise due to run time error"""
        self.assertRaises(ZeroDivisionError, divide_by(5,3))

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

UNIT TESTING

TEST OUTCOMES AND COMPARED TO EXPECTED

```
# simple_test.py

import unittest
import simple

class TestSimple(unittest.TestCase):

    def test_is_true(self):
        self.assertEqual(simple.always_returns_true(),True)

    def test_is_even(self):
        self.assertEqual(simple.is_even(2),True)
        self.assertNotEqual(simple.is_even(3),True)

    def test_divide_by(self):
        """Test exception raise due to run time error"""
        self.assertRaises(ZeroDivisionError, divide_by(5,0))

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

UNIT TESTING

LET PROGRAM BE CALLED STANDALONE

```
# simple_test.py

import unittest
import simple

class TestSimple(unittest.TestCase):

    def test_is_true(self):
        self.assertEqual(simple.always_returns_true(),True)

    def test_is_even(self):
        self.assertEqual(simple.is_even(2),True)
        self.assertNotEqual(simple.is_even(3),True)

    def test_divide_by(self):
        """Test exception raise due to run time error"""
        self.assertRaises(ZeroDivisionError, divide_by,5,3)

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

UNIT TESTING

```
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

UNIT TESTING

```
class TestSimple(unittest.TestCase):
    def test_is_true(self):
        self.assertEqual(simple.is_true(),False)

======================================================================
FAIL: test_is_true (__main__.TestSimple)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "simple_test.py", line 11, in test_is_true
    self.assertEqual(simple.is_true(),False)
AssertionError: True != False
Ran 1 test in 0.000s

FAILED (failures=1)
```

UNIT TESTING

- Assertions

Method	Checks that
assertEqual(a, b)	`a == b`
assertNotEqual(a, b)	`a != b`
assertTrue(x)	`bool(x) is True`
assertFalse(x)	`bool(x) is False`
assertIs(a, b)	`a is b`
assertIsNot(a, b)	`a is not b`
assertIsNone(x)	`x is None`
assertIsNotNone(x)	`x is not None`
assertIn(a, b)	`a in b`
assertNotIn(a, b)	`a not in b`
assertIsInstance(a, b)	`isinstance(a, b)`
assertNotIsInstance(a, b)	`not isinstance(a, b)`

UNIT TESTING

WE NEED TO PROGRAM FAILURES AND EDGE CASES INTO OUR TESTS

```
def test_divide_by(self):
    """Test exception raise due to run time error"""
    self.assertRaises(ZeroDivisionError, divide_by,5,3)
    self.assertEqual(divide_by(6,3),2)
```

MODULES

- Writing code to test may require you to refactor in a less efficient manner

```
def compute_tax_and_tip(amount):
    return ((amount + \
             (amount*.0875)) * 1.20)


# Refactored for reusability and
# testing
def compute_tax_and_tip(amount):
    tax = compute_tax(amount)
    tip = compute_tip(amount + tax,20)
    return amount + tax + tip

def compute_tax(amount):
    pass

def compute_tip(amount):
    pass
```

UNIT TESTING

- Unit test can be a useful tool but they can be abused
- In practice, they are used for larger codebases and applications
- You can be a successful, careful programmer without them, but they can help if used effectively

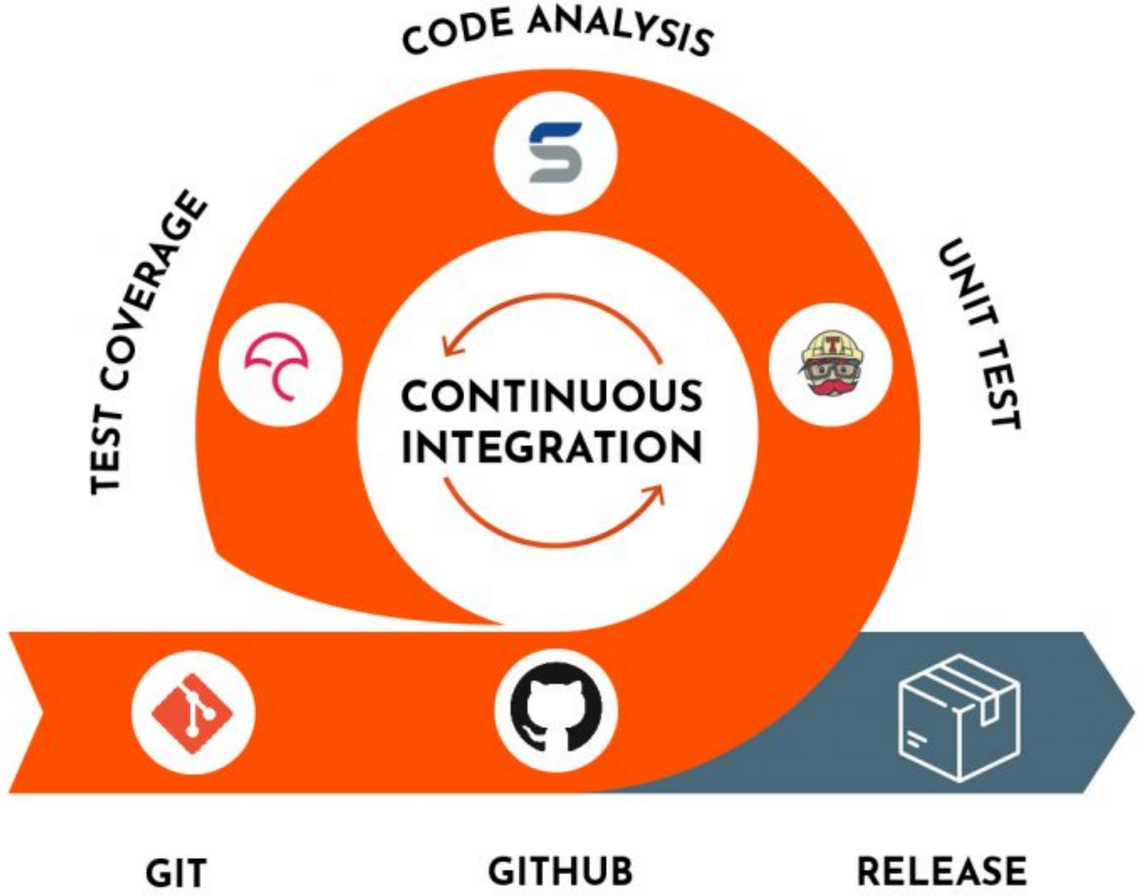
CONTINUOUS INTEGRATION WITH GITHUB

MODULE 5
MPCS 50101

THE UNIVERSITY OF CHICAGO

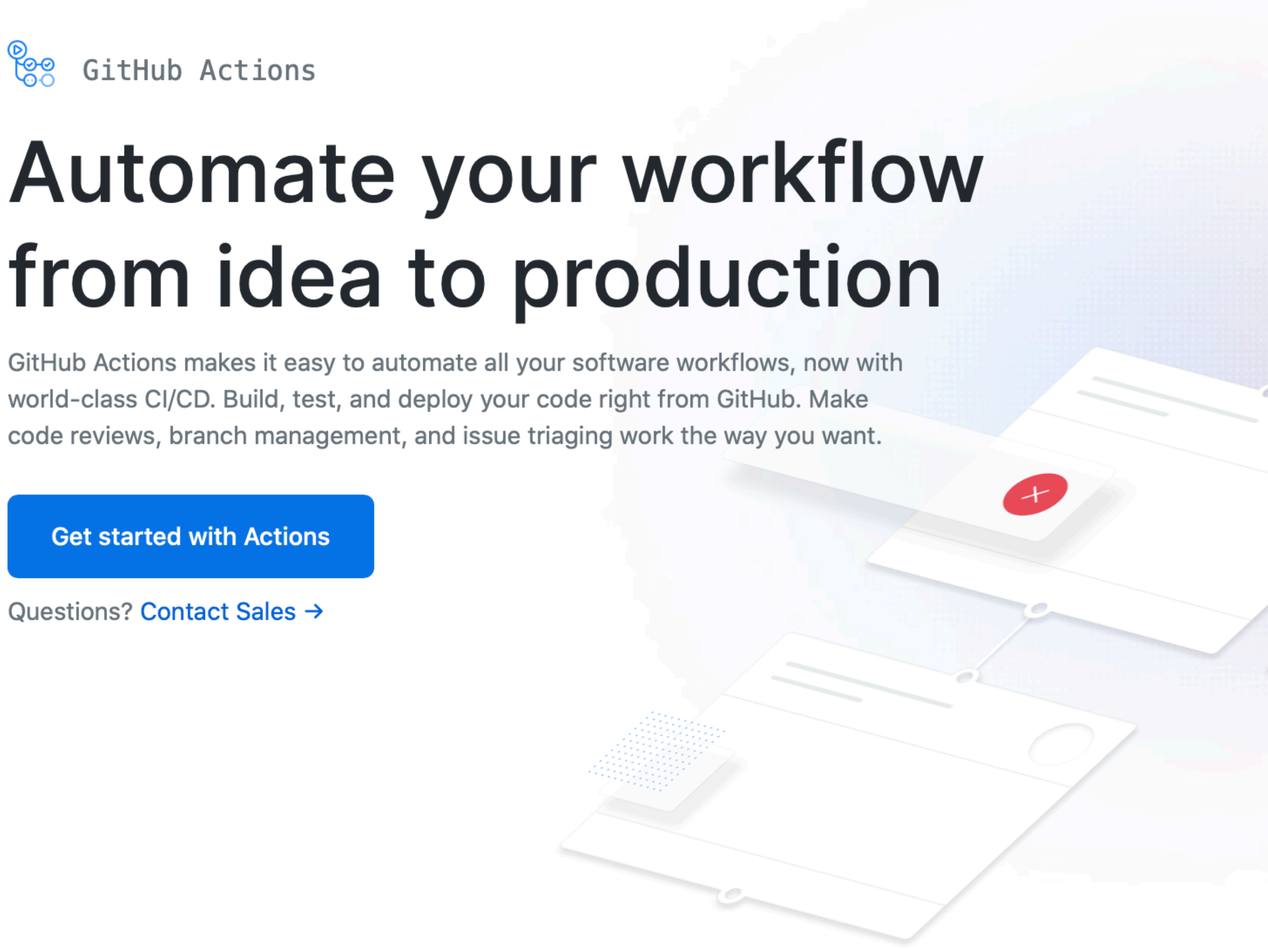
CI WITH GITHUB

- Continuous Integration (CI) is a development practice
  - Developers integrate code into a shared repository frequently
  - Each integration can then be verified by an automated build and automated tests



CI WITH GITHUB

- GitHub Actions provides support for CI
  - Supports testing and deployment
  - Hooks to "do stuff" at different stages



THE END

MODULE 5
MPCS 50101

THE UNIVERSITY OF CHICAGO