Practical Coding in Python

Learn to write and validate your own code

Darren Kessner, PhD

(revised September 1, 2025)

Contents

Introduction

This book is a practical guide for learning to write code in Python. The structure follows Practical Coding (Java), and in fact the exercises are the same or very similar.

This is meant as a guidebook for self-study, so that you can learn how to use Python to solve the same problems that you can solve with Java.

I emphasize writing unit tests to validate your own code. Writing tests for your own code will give you confidence that your code does what you expect it to. With the development of generative AI, it is easy to generate code to solve any problem in any programming language, and the code may *seem* to work. However, it is now even more important for software developers to ensure that code behaves as expected.

Rather than *reading* this book, I recommend that you focus on *writing code*, i.e. doing the exercises for each chapter. The only way to learn to write code is to actually write code. Each chapter has working code examples that illustrate new syntax or concepts, together with the output from running the code. You will learn much more by struggling with and completing the exercises than by reading the examples.

About the book

This book is based on notes, demo code, and coding exercises I have written and used over the past 10 years of teaching AP Computer Science at Marlborough School in Los Angeles.

Various versions of this content have been published previously on my class webpages with open source licensing,

About the author

I am the Program Head of Computer Science and Software Innovation at Marlborough School in Los Angeles, where I have taught Math and Computer Science for 11 years.

I am also a software developer with over 25 years of experience writing software in a wide variety of fields, including computer security, computer graphics, and scientific applications. My published academic papers include contributions to the areas of bioinformatics, proteomics, and population genetics.

I am a strong proponent of free and open source software, open public data, and open educational resources. I am also an advocate for increasing the diversity of voices in the STEM fields in general, and in software development in particular.

In the classroom I use free and open source software, open public data, and open educational resources.

Darren Kessner, PhD
https://dkessner.github.io

0. Hello, world!

This zeroth chapter is a quick overview of Python syntax.

If you have experience coding in a Java-like language (C/C++, Processing, Arduino), pay extra attention to the differences between Java and Python, which I've noted in the comments of the code examples.

One of your main goals this chapter is to write a FizzBuzz program (see Coding Exercises).

Hello

```
#
# hello.py
#

print("Hello, world!")
```

Output:

```
Hello, world!
```

Basics

```
#
# basics.py
#

# Comments are specified by the octothorpe (hashtag) `#`.

# You declare a variable by assigning it.

# In Python, the variable's type is dynamic: both the type and
# value of a variable can change.  All types are reference types:
# every variable refers to an object, but the class can change.
# The basic types in Python are similar to the wrapper types
# Integer, Float, etc. in Java.

x = 5       # integer
print("x:", x)

x = 1.23    # float
print("x:", x)

x = "hello" # string
print("x:", x)

x = True    # boolean
print("x:", x)
```

Output:

```
x: 5
x: 1.23
x: hello
x: True
```

Loops

```
#
# loops.py
#


# Lists are declared with the square brackets, and may contain
# objects with different types.

things = [7, 4.20, "juggling ball", True]
print("things:", things)

# List indexing is 0-based, just like Java.

print("things[0]:", things[0])
print("things[2]:", things[2])

# You can also give index ranges, resulting in a "slice" of the
# list, and negative indices which count from the end.

print("things[0:2]", things[0:2])
print("things[-1]:", things[-1])
print("things[-2]:", things[-2])

# The Python for loop to iterate through a list is similar to the
# Java for-each loop.

# Important difference from Java: Python uses the colon followed by
# indented code to indicate scope, where Java uses the curly braces
# {}.  You can indent any amount, but you must be consistent within
# the scope.

print("printing items")
for item in things:
    print("item:", item)

# You can also use the range() function to iterate through a range
# of integers.  range(begin, end) returns the integers in the
# half-open interval [begin,end). range(end) is shorthand for
# range(0, end).  You can also specify a step parameter:
```

```
# range(begin, end, step).

print("printing range(5)")
for x in range(5):
    print("x:", x)

print("printing range(11, 20, 2)")
for x in range(11, 20, 2):
    print("x:", x)


# Conditions are checked with `if`, with scope specified by
# indentation as with `for`.

# Note that = and == are the similar to Java:
#   =  is assignment
#   == is comparison (returns a boolean)

# However, Python is different from Java in that == compares
# values, not references.

print("printing with conditions")
for i in range(10):
    if i%2 == 0:
        print("Even")
    elif i == 7:                # "else if" in Java
        print("Lucky")
    else:
        print(i)
```

Output:

```
things: [7, 4.2, 'juggling ball', True]
things[0]: 7
things[2]: juggling ball
things[0:2] [7, 4.2]
things[-1]: True
things[-2]: juggling ball
printing items
item: 7
```

```
item: 4.2
item: juggling ball
item: True
printing range(5)
x: 0
x: 1
x: 2
x: 3
x: 4
printing range(11, 20, 2)
x: 11
x: 13
x: 15
x: 17
x: 19
printing with conditions
Even
1
Even
3
Even
5
Even
Lucky
Even
9
```

Sequences

```
#
# sequences.py
#

# Print multiples of 7

print("Multiples of 7")
for i in range(30):
    if i%7 == 0:
        print(i)

# Print multiples of 7 again using range(begin, end, step)

print("Multiples of 7")
for i in range(0, 29, 7):
    print(i)


# Arithmetic sequences have a common difference.  For example, the
# sequence {3, 10, 17, 24, ...} has common difference 7.

# Print an arithmentic sequence using its recursive formula.

print("Arithmetic sequence: recursive")
value = 3
for i in range(5):
    print(value)
    value += 7

# Print an arithmetic sequence using its explicit formula.

print("Arithmetic sequence: explicit")
for i in range(5):
    print(3 + i*7)
```

Output:

```
Multiples of 7
0
```

```
7
14
21
28
Multiples of 7
0
7
14
21
28
Arithmetic sequence: recursive
3
10
17
24
31
Arithmetic sequence: explicit
3
10
17
24
31
```

Coding Exercises: Hello

1. Multiples of 3

Write a program that prints the first 10 multiples of 3.

2. FizzBuzz

Write a FizzBuzz program.

Your program should iterate through the first 30 positive integers, printing each one. However, if the integer n is a multiple of 3, print `Fizz` instead of the number. And if n is a multiple of 5, print `Buzz` instead. And if n is a multiple of both 3 and 5, print `FizzBuzz` instead.

Sample output:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
.
.
.
```

3. Geometric sequence

Write a program that prints out the first terms of a geometric sequence, i.e. a sequence with a common ratio, for example: 3, 6, 12, 24, 48, ...

4. Cubes

Write a program that prints out the cubes of the counting numbers: 0, 1, 8, 27, 64, 125, ...

5. Fibonacci sequence

Write a program that prints out the first 30 terms of the Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Hint: I find it easiest to think about this problem using 3 variables: a and b slide up the sequence, and we use a temporary variable c to help do this.

Challenge: After you've done this exercise, try doing it using only 2 variables.

Challenge: Try printing out the ratios of successive terms of the Fibonacci sequence. The sequence of ratios approaches a limit - do you recognize what this limit is?

1. Functions and Testing

In this chapter your goal is to become familiar with functions, and writing unit tests to test your functions. This is the most important chapter of the book.

Unit tests are low-level tests of a single function (the target function). In the unit test function, you are given sample input and the expected output (return value). The unit test function runs the target function with the given input and checks that the return value matches what was expected.

Unit test functions are important for verifying the behavior of your code. As you add more code to your project, it is easy to introduce new bugs. Running the unit tests after any changes will give you confidence that your functions still behave as you expect.

Unit tests are also important for ongoing maintenance of your code. For example, if you find a new bug, you can:

- write a new unit test that fails due to the bug
- fix the bug
- verify that all unit tests pass

In the software development world, this procedure of writing tests first is called *test driven development (TDD)*.

Functions

```
#
# functions.py
#


def hello():
    print("Hello, world!")

hello()

# Difference from Java: note the lack of type specifications in
# either input parameters or return values

def sum(a,b):
    return a+b

print("sum(5,7):", sum(5,7))

# dynamic typing: for strings, + is concatenation
print(sum("Hello, ", "world!"))


def is_odd(n):
    return n%2 == 1

print("is_odd(5):", is_odd(5))
print("is_odd(7):", is_odd(7))
print("is_odd(8):", is_odd(8))


# Python has a very useful feature called List Comprehensions.

odd_numbers = [i for i in range(10) if is_odd(i)]
print("odd_numbers:", odd_numbers)

even_numbers = [2*i for i in range(5)]
print("even_numbers:", even_numbers)
```

```
# In general, you can write:
# my_list = [f(x) for x in things if P(x)]
#
# You can think of P() as a condition that gives you a subset of
# things, and f() is a transformation of each thing x.
```

Output:

```
Hello, world!
sum(5,7): 12
Hello, world!
is_odd(5): True
is_odd(7): True
is_odd(8): False
odd_numbers: [1, 3, 5, 7, 9]
even_numbers: [0, 2, 4, 6, 8]
```

Monkey trouble

```
#
# monkey_trouble.py
#

# This exercise is from CodingBat by Nick Parlante
# https://codingbat.com/prob/p181646

# We have two monkeys, a and b, and the parameters aSmile and
# bSmile indicate if each is smiling. We are in trouble if they are
# both smiling or if neither of them is smiling. Return true if we
# are in trouble.

# monkey_trouble() is the function you want to test.

def monkey_trouble(aSmile, bSmile):
    return aSmile == bSmile

# test_monkey_trouble() is the unit test function, with arguments
# for given input and the expected output.

def test_monkey_trouble(aSmile, bSmile, expected):
    result = monkey_trouble(aSmile, bSmile)

    print(f"aSmile: {aSmile}, bSmile: {bSmile}, "
          f"expected: {expected}, result: {result}")

    if result == expected:
        print("PASSED")
    else:
        print("FAILED")


# We run several tests, varying the input parameters and checking
# that we get the output we expect from monkey_trouble().

test_monkey_trouble(True, True, True)
test_monkey_trouble(False, False, True)
test_monkey_trouble(True, False, False)
```

```
# Named arguments can help the readability of your code.

test_monkey_trouble(aSmile=False, bSmile=True, expected=False)
```

Output:

```
aSmile: True, bSmile: True, expected: True, result: True
PASSED
aSmile: False, bSmile: False, expected: True, result: True
PASSED
aSmile: True, bSmile: False, expected: False, result: False
PASSED
aSmile: False, bSmile: True, expected: False, result: False
PASSED
```

Coding Exercises: Functions and Testing

1. Vampire

A person is a vampire if she is asleep during waking hours (6:00 to 22:00), or awake during sleeping hours (before 6:00 or after 22:00). Write a function `is_vampire(hour, awake)` where `hour` is the time represented as a `float` (e.g. 6.5 means 6:30), and `awake` represents whether the person is awake (`True` or `False`), returning `True` if that person is a vampire and `False` otherwise. Most imporantly, write a unit test function and several unit tests.

2. Good Deal

A store has marked down the prices of many items, but you only want to buy something if the discount is more than 25% (or in other words, the sale price is $< 75\%$ of the original price). Write a function `good_deal(originalPrice, salePrice)` that returns true if you're getting a good deal on the item. Most importantly, write a unit test function and several unit tests.

3. (Challenge) Prime Numbers

Write a program to print the prime numbers.

To do this, first write a function `is_prime()`:

```
def is_prime(n):
{
    // return True <-> n is prime
}
```

Write a unit test function and several unit tests for this function.

Then in your program, loop through the first 100 integers and print only the ones for which `is_prime()` returns `True`.

2. Strings and Math

This chapter demonstrates the use of strings and math functions in Python.

Hello strings

```
#
# hello_strings.py
#

hello = "HelloWorld"
print("hello: ", hello)
print("len(hello): ", len(hello))

# String indexing is 0-based, and you create substrings with the
# slice notation (like lists).

print("hello[0]: ", hello[0])
print("hello[0:5]:", hello[0:5])
print("hello[5:]: ",  hello[5:])
print("hello[5:-2]: ", hello[5:-2])

# f-strings can be useful for creating strings that contain Python
# expressions

print(f"hello: {hello} {len(hello)} {len(hello) == 10}")
```

Output:

```
hello:  HelloWorld
len(hello):  10
hello[0]:  H
hello[0:5]: Hello
hello[5:]:  World
hello[5:-2]:  Wor
hello: HelloWorld 10 True
```

Hello math

```
#
# hello_math.py
#

# Importing the math module gives you access to mathematical
# constants and functions.

import math

print("math.pi:", math.pi)
print("math.cos(math.pi):", math.cos(math.pi))

# You can import all the names (constants and functions) from a
# module for convenience.  In larger projects it is not generally
# recommended to do this due to the potential for name conflicts
# between functions in different modules.

from math import *

print("pi:", pi)
print("e:", e)

print("cos(0):", cos(0))
print("sin(0):", sin(0))
print("cos(pi):", cos(pi))
print("sin(pi):", sin(pi))

result = sin(pi)

if abs(result) < 1e6:
    print("Yay!")
else:
    print("Boo!")
```

Output:

```
math.pi: 3.141592653589793
math.cos(math.pi): -1.0
pi: 3.141592653589793
e: 2.718281828459045
```

```
cos(0): 1.0
sin(0): 0.0
cos(pi): -1.0
sin(pi): 1.2246467991473532e-16
Yay!
```

Hello random

```
#
# hello_random.py
#

import random

# random() returns a value in [0,1)

print("Random values in [0,1)")
for i in range(5):
    print(random.random())


# uniform(a,b) returns a value in [a,b)

print()
print("Random values in [0,100)")
for i in range(5):
    print(random.uniform(0, 100))


# randint(a,b) returns an integer in [a,b] (note: closed interval)

print()
print("Random integers in [1,10]")
for i in range(5):
    print(random.randint(1, 10))
```

Output:

```
Random values in [0,1)
0.585961360088896
0.18310505104513708
0.42122895197406385
0.4586921921900672
0.8552122193463879

Random values in [0,100)
0.12792921124947787
```

```
12.627624801719916
61.359384626313584
15.253554451169215
78.34393981946957

Random integers in [1,10]
3
3
1
6
6
```

Coding Exercises: Strings and Math

1. Greetings.

Write a function `greetings()` that takes a single String `name` and returns returns a greeting using the given name. Be sure to include unit tests.

Sample output:

```
greetings("Dr. Kessner") -> "Hello, Dr. Kessner, how are you?"
greetings("Ascii Cat") -> "Hello, Ascii Cat, how are you?"
greetings("Sydneys") -> "Hello, Sydneys, how are you?"
```

2. Attention.

Write a function `attention()` that takes a single `String` as input and returns `true` if the string starts with "Hey you!". Be sure to include unit tests.

Sample output:

```
attention("Hello, my name is Inigo Montoya.") -> false
attention("Excuse me, Dr. Kessner?") -> false
attention("Hey you! Give me your code!" -> true
```

3. Coin flip.

Write a function that flips a coin randomly, returning a String, either "Heads" or "Tails". Functions involving randomness are a little tricky to write unit tests for. So you should just have your `main()` function print the results from 10 or 20 coin flips to try out your function.

4. Die rolling

Write a function that returns the result of rolling a single 6-sided die. In other words, when you call the function, it should randomly return 1, 2, 3, 4, 5, or 6.

3. Loops & Algorithms

In this chapter we discuss the different loop constructs that you can use in Python. We also practice implementing algorithms that use a loop to calcluate a result.

Hello loops

```
#
# hello_loops.py
#


# for loop

things = [1.23, 666, "juggling balls"]

print("things:")
for thing in things:
    print(thing)


# while loop

print()
print("while loop")

value = 0

while value < 5:
    print(value)
    value += 1


# continue and break


print()
print("continue and break")

value = 0

while True:              # loop forever
    value += 1
    if value%2 == 0:     # even: do nothing
        continue
    print(value)
```

```
    if value > 5:       # break out of loop
        break
```

Output:

```
things:
1.23
666
juggling balls

while loop
0
1
2
3
4

continue and break
1
3
5
7
```

Hello algorithms

```
#
# hello_algorithms.py
#


# In each of these examples, we use a loop to perform a
# calculation.


# Find the first 'e' character in s.

def find_E(s):
    for i in range(len(s)):
        if s[i] == 'e':
            return i

s = "Dr. Kessner"
print("s: ", s)

print("find_E(s):", find_E(s));


# Count the number of 'e' characters in s.

def count_E(s):
    total = 0
    for c in s:
        if c == 'e':
            total += 1
    return total


print("count_E(s):", count_E(s));


# Calculate sum of integers from 1 to n.

def sum(n):
    total = 0
```

```
    for i in range(n+1):
        total += i
    return total


print()
print("sum(3):", sum(3))
print("sum(4):", sum(4))
print("sum(5):", sum(5))
```

Output:

```
s:  Dr. Kessner
find_E(s): 5
count_E(s): 2

sum(3): 6
sum(4): 10
sum(5): 15
```

Binimate

```
#
# binimate.py
#


# Full example with unit tests


# binimate(s): kill every other character and return the result

def binimate(s):
    result = ""
    for i in range(len(s)):
        if i%2==0:
            result += s[i]
    return result


# unit test function: run the function binimate() and verify the
# output is what you expect

def test_binimate(s, expected):
    result = binimate(s)
    print("s:", s, "expected:", expected, "result:", result)
    if result == expected:
        print("Woohoo!")
    else:
        print("Boohoo!")


# multiple unit tests

test_binimate("Dr. Kessner", "D.Ksnr");
test_binimate("Briley", "Bie");
test_binimate("Jasmine", "Jsie");
test_binimate("Sophia", "Spi");
```

Output:

```
s: Dr. Kessner expected: D.Ksnr result: D.Ksnr
Woohoo!
s: Briley expected: Bie result: Bie
Woohoo!
s: Jasmine expected: Jsie result: Jsie
Woohoo!
s: Sophia expected: Spi result: Spi
Woohoo!
```

Coding Exercises: Loops and Algorithms

Implement the following functions, including unit tests. A few example tests are shown. Add at least 2 more tests of your own for each function.

Sum of Squares

```
sum_of_squares(1) -> 1
sum_of_squares(2) -> 1+4 = 5
sum_of_squares(3) -> 1+4+9 = 14
```

Count Occurences

```
count_occurrences("Mississippi", "iss") -> 2
count_occurrences("bananananana", "na") -> 4
```

Reverse String

```
reverse("bad") -> "dab"
reverse("Hello, world!") -> "!dlrow ,olleH"
reverse("tacocat") -> "tacocat"
```

Factorial

```
factorial(0) -> 1
factorial(1) -> 1
factorial(2) -> 2*1 = 2
factorial(3) -> 3*2*1 = 6
factorial(4) -> 4*3*2*1 = 24
factorial(5) -> 5*4*3*2*1 = 120
```

Interlace Two Strings

```
interlace("abc", "123") -> "a1b2c3"
interlace("bed", "ras") -> "breads"
```

Find 2nd “a”

```
find_2nd("banana") -> 3
find_2nd("happy birthday") -> 12
```

Add “na” suffix

```
add_na(0) -> "ba"
add_na(1) -> "bana"
add_na(2) -> "banana"
add_na(3) -> "bananana"
```

Calculate Sum of Powers of 2

```
sum_powers(0) -> 0
sum_powers(1) -> 0+1 = 1
sum_powers(2) -> 0+1+2 = 3
sum_powers(3) -> 0+1+2+4 = 7
sum_powers(4) -> 0+1+2+4+8 = 15
```

Appendix A: Virtual Environments and Libraries

Python 3 has a module called `venv` for creating virtual environments for your projects.

A virtual environment is an independent Python installation at a location (subdirectory) that you specify. When you *activate* a virtual environment, that particular Python installation will be used to run your programs.

If your project depends on external libraries, you can install these into your virtual environment using the standard Python installation tool `pip`.

This allows you to:

1) Keep each software project in a separate environment.

2) Specify your project's dependencies precisely.

Here's a quick start.

Create a new virtual environment

This creates the directory `venv_name`.

```
python3 -m venv venv_name
```

Often people use the name `venv` for the installation directory.

Note: You do not want to include your virtual environment installation directory in your repository. This folder will contain a large number of binary files.

Activate the virtual environment

The new directory `venv_name` has a sub-directory `bin`, in which there is a script called `activate`.

```
venv_name
|-- bin
    |- activate
```

Calling this script with the `source` command activates the virtual environment by setting your `PATH` and other environment variables.

```
source venv_name/bin/activate
```

This will also change your prompt to include `(venv_name)`, to indicate that the virtual environment is active.

Deactivate the virtual environment

```
deactivate
```

Install requirements into the virtual environment

First activate the virtual environment. Then calling `pip install` will install stuff into the active virtual environment.

```
source venv_name/bin/activate
pip install library_name
```

Specfiy your dependencies in requirements.txt

It is common practice to list your dependencies in a file `requirements.txt` in the root directory of your project:

```
matplotlib
numpy
```

Including `requirements.txt` in your source code repository allows you to easily reproduce your development environment in another location.

Use the `-r` flag with `pip install` to install the libraries listed in `requirements.txt`.

```
pip install -r requirements.txt
```

Set up your dev environment in another location

The `requirements.txt` file helps you set up a new development environment (for example to work on your project on another computer, or to work with a collaborator).

```
python3 -m venv venv_name
source venv_name/bin/activate
pip install -r requirements.txt
```

Exercises: Installing and using libraries

In these exercises you will install and try out various Python libraries. For each example:

1. Create a new folder for your project.
2. Create a file in this folder called `requirements.txt` to specify the libraries you want to install, one library per line:

   ```
   emoji
   ```

3. Create a new virtual environment called `venv`.

   ```
   python3 -m venv venv
   ```

4. Activate the virtual environment.

   ```
   source venv/bin/activate
   ```

5. Install the libraries listed in `requirements.txt`.

   ```
   pip install -r requirements.txt
   ```

6. Copy and run the example code.

Hello emoji

```
#
# hello_emoji.py
#


# Installation: 'emoji' in requirements.txt


from emoji import *


names = [
    ':robot:',
    ':pile_of_poo:',
    ':cat:',
    ':cat_face:',
    ':grinning_cat:',
    ':grinning_face:',
    ':upside-down_face:',
    ':nerd_face:',
    ':skull_and_crossbones:',
    ':person_cartwheeling:',
    ':person_juggling:',
    ':person_in_lotus_position:',
    ':circus_tent:',
    ':roller_skate:',
]

for name in names:
    print(name, emojize(name))
```


:robot:
:pile_of_poo:
:cat:
:cat_face:
:grinning_cat:
:grinning_face:
:upside-down_face:
:nerd_face:
:skull_and_crossbones:
:person_cartwheeling:
:person_juggling:
:person_in_lotus_position:
:circus_tent:
:roller_skate:

Hello requests

```
#
# hello_requests.py
#


# Installation: 'requests' in requirements.txt


import requests


# The `requests` library allows you to send HTTP requests, for
# accessing web pages or web APIs.
# https://requests.readthedocs.io

# Weather API documentation
# https://www.weather.gov/documentation/services-web-api

# National Weather Service (NWS)
# https://www.weather.gov/

# NOAA
# https://www.noaa.gov/


def main():

    # https://api.weather.gov/points/{latitude},{longitude}
    url = "https://api.weather.gov/points/34.0699142,-118.3294098"

    # Use the `get()` function to retrieve a webpage (GET request).

    response = requests.get(url)

    # Status code 200 is success.

    print("status_code:", response.status_code)
    print()
```

```
    if response.status_code != 200:
        print("Error.")
        return

    # The response from a webpage will generatlly be HTML.  The
    # response from a web API will usually be in JSON format, which
    # you can think of as a bunch of nested dictionaries.

    result_json = response.json()
    properties = result_json['properties']

    city = properties['relativeLocation']['properties']['city']
    forecast_url = properties['forecast']

    print("city:", city)
    print("forecast_url:", forecast_url)
    print()

    # Accessing the actual forecast requires a second GET request.

    response = requests.get(forecast_url)

    properties = response.json()['properties']
    periods = properties['periods']

    for period in periods:
        print(period['name'])
        print(period['detailedForecast'])
        print()


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

Output:

```
status_code: 200

city: West Hollywood
forecast_url: https://api.weather.gov/gridpoints/LOX/152,46/forecast
```

```
Today
Patchy fog before 11am. Mostly sunny, with a high near 80.
Southwest wind 5 to 10 mph.

Tonight
Patchy fog after 5am. Partly cloudy, with a low around 64. South
wind 0 to 10 mph.

Saturday
Patchy fog before 11am. Mostly sunny, with a high near 80. South
southwest wind 0 to 10 mph.

Saturday Night
Patchy fog after 11pm. Partly cloudy, with a low around 64. South
wind 0 to 10 mph.

Sunday
Patchy fog. Partly sunny, with a high near 82. South southwest wind
0 to 10 mph.

Sunday Night
Patchy fog before 11pm. Partly cloudy, with a low around 63.

Monday
Patchy fog after 5pm. Mostly sunny, with a high near 78.

Monday Night
Patchy fog. Cloudy, with a low around 63.

Tuesday
Patchy fog before 11am. Mostly sunny, with a high near 77.

Tuesday Night
Patchy fog after 11pm. Partly cloudy, with a low around 62.

Wednesday
Patchy fog before 11am. Mostly sunny, with a high near 76.

Wednesday Night
```

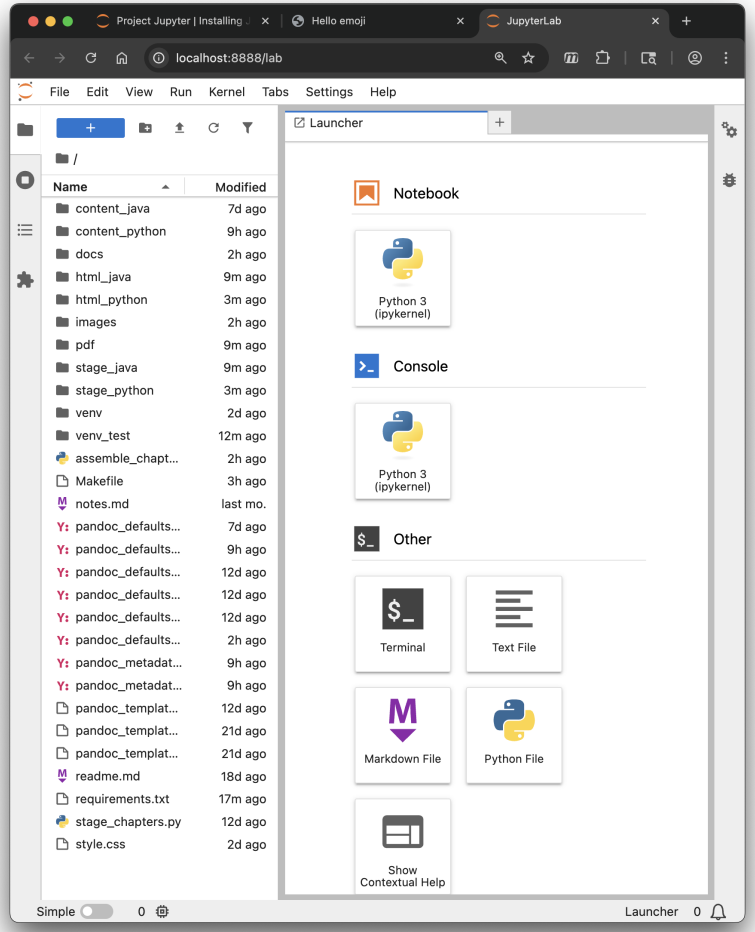
```
Partly cloudy, with a low around 61.

Thursday
Mostly sunny, with a high near 76.

Thursday Night
Partly cloudy, with a low around 61.
```

Hello, Juypter Lab!

1. Create your virtual environment, and install `jupyterlab` via your `requirements.txt` file.
2. On the command line, start the Juypter server with the command `jupyter lab`. Stop the server with `ctrl-c`.
3. Your browser should open to a Jupyter Lab page.



Hello py5

py5 is Processing for Python.
https://py5coding.org/

This is the "module" mode hello program.

Information about the different py5 modes here:
https://py5coding.org/content/py5_modes.html

```
#
# hello_py5.py
#

import py5

def setup():
    py5.size(300, 200)
    py5.rect_mode(py5.CENTER)

def draw():
    py5.rect(py5.mouse_x, py5.mouse_y, 10, 10)

py5.run_sketch()
```