PYTHON

QuickStart Guide®

AUDIOBOOK COMPANION

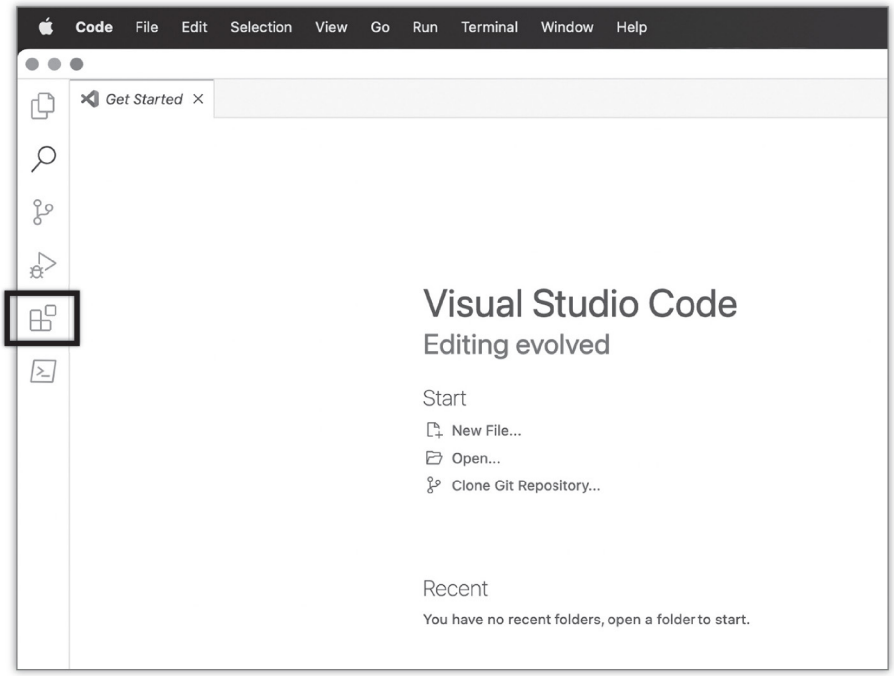
QuickStart Guides

WWW.QUICKSTARTGUIDES.COM

TABLE OF CONTENTS

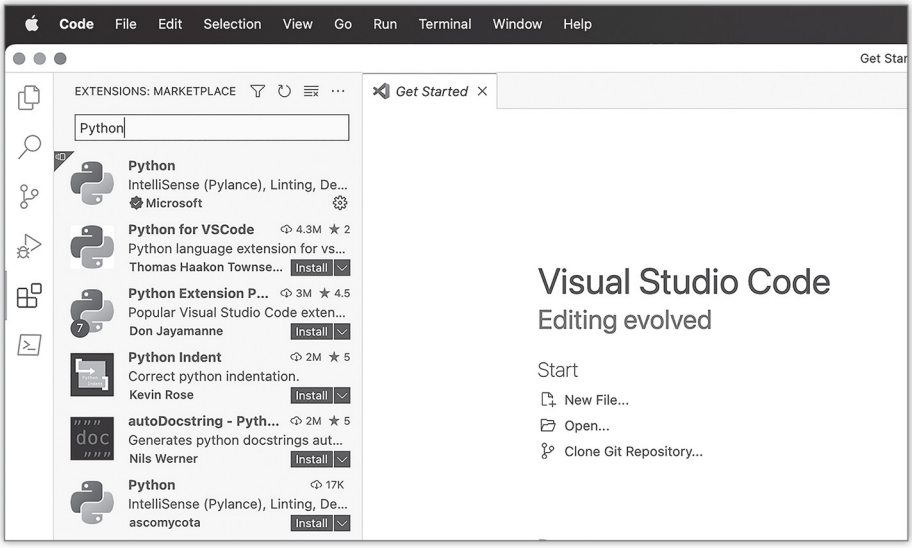
Introduction

fig. 1



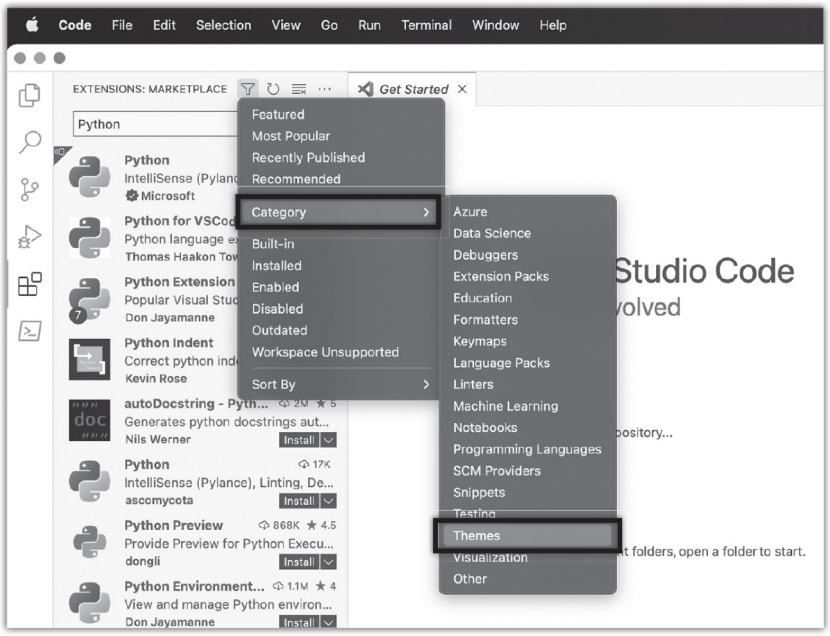
The Visual Studio Code welcome screen.
The Extensions icon is highlighted in this figure on the left-hand pane.

fig. 2



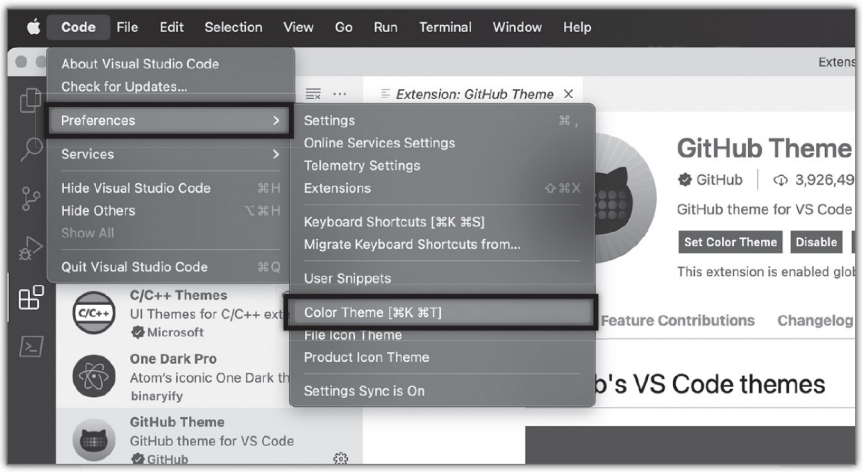
Searching for the Python extension in the extensions list of Visual Studio Code.
In this figure, the official Python extension is shown with a check mark badge and the publisher, Microsoft, beneath the listing.

fig. 3



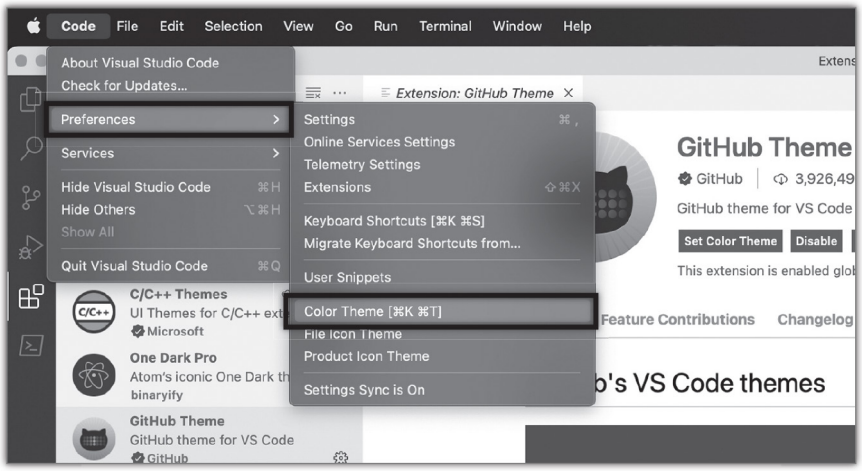
Accessing the Themes category filter in the extension viewer.

fig. 4



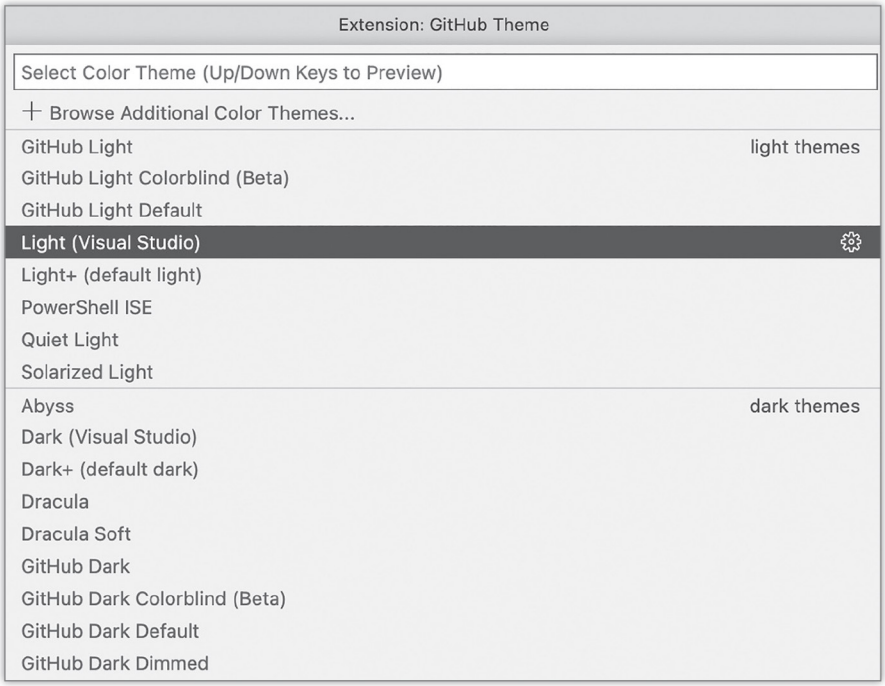
The Color Theme menu item in the *Code* > *Preferences* menu.

fig. 4



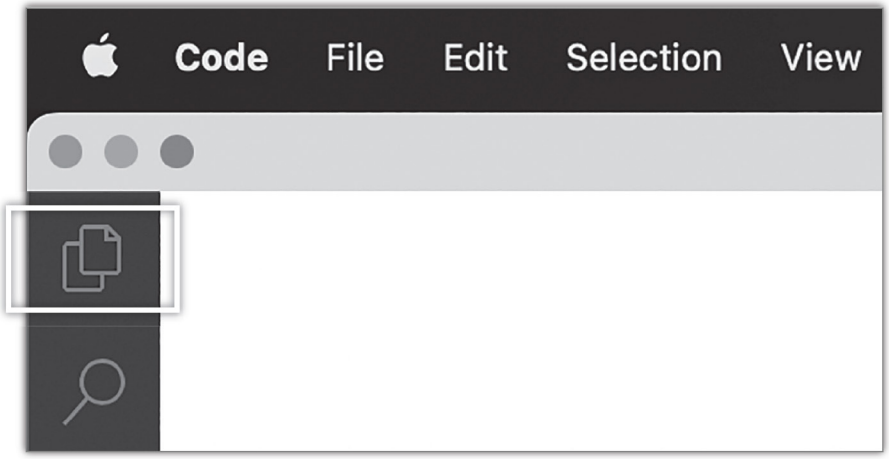
The Color Theme menu item in the *Code* > *Preferences* menu.

fig. 5



The color theme selector.

fig. 6

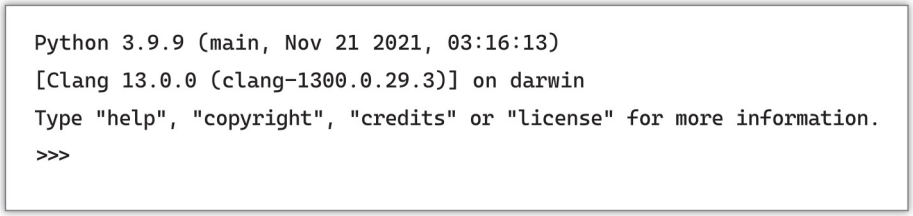


The Explorer icon is in the box.

Linux Installation Commands:

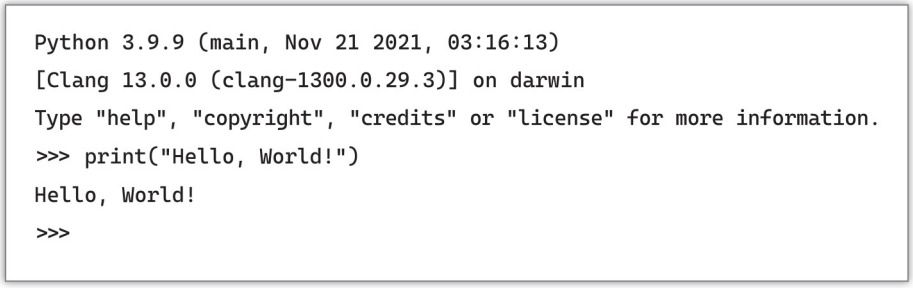
```
sudo apt install python3
sudo yum install python3
sudo dnf install python3
sudo pacman -S python
```

fig. 7



The Python interpreter prompt.

fig. 8

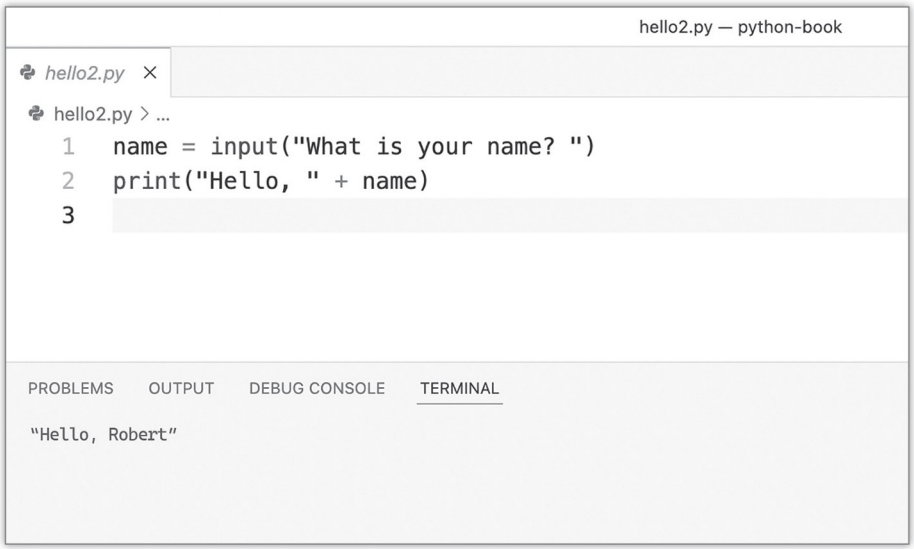


The Python interpreter after running the "Hello World!" print statement

fig. 9

The Visual Studio Code display after running your Python program.
Note the "Hello World!" output in the bottom pane of the window.

fig. 10



The Visual Studio Code display after running hello2.py.
Note the "Hello, Robert" output in the bottom pane of the window.

fig. 11

```
word1 = "Hello"
word2 = "World!"
print(word1 + " " + word2)
```

The inline string operation is circled in figure 11.

fig. 12

```
1       price = input("What is the price of a cup of coffee? ")
2       cups = input("How many cups do you want? ")
3       total = price * cups

Exception has occurred: TypeError x
can't multiply sequence by non-int of type 'str'
     File "/Users/rwoliver2/Source/cups.py", line 3, in <module>
          total = price * cups

4       print("Your total is $" + total + " for " + cups + " cups.")
5
```

Something went wrong! A TypeError has occurred.

Code Line 1:

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

Code Line #2:

```
name = input("What is your name? ")
```

Code Line #3:

```
name = input("What is your name? ")
print("Hello, " + name)
```

Code Line #4:

```
word1 = "Hello"
word2 = "World!"
print(word1 + word2)
```

Code Line #5:

```
word1 = "Hello "
word2 = "World!"
print(word1 + word2)
```

Code Line #6:

```
word1 = "Hello"
word2 = "World!"
print(word1 + " " + word2)
```

Code Line #7:

```
word1 = "Hello"
word2 = "World!"
space = " "
print(word1 + space + word2)
```

Code Line #8:

```
greeting = "Well, hello there!"
hello = greeting[6:11]
print(hello)
```

Code Line #9:

```
greeting = "Well, hello there!"
print(greeting[6:11])
```

Code Line #10:

```
greeting = "Well, hello there!"
print(greeting[6:])
```

Code Line #11:

```
greeting = "Well, hello there!"
print(greeting[:4])
```

Code Line #12:

```
apples = 4
pears = 8
attendees = 3042
zip_code = 12345
```

Code Line #13:

```
price = 4.95
gpa = 3.74
pi = 3.14159
```

Code Line #14:

```
price = input("What is the price of a cup of coffee? ")
cups = input("How many cups do you want? ")
total = price * cups
print("Your total is $" + total + " for " + cups + " cups.")
```

Code Line #15:

```
total = price * int(cups)
```

Code Line #16:

```
total = float(price) * int(cups)
```

Code Line #17:

```
print("Your total is $" + str(total) + " for " + cups + " cups.")
```

Code Line #18:

```
What is the price of a cup of coffee? 3.99
How many cups do you want? 3
Your total is $11.97 for 3 cups.
```

Code Line #19:

```
number_of_cups = int(cups)
actual_price = float(price)
```

Code Line #20:

```
total = actual_price * number_of_cups
```

Code Line #21:

```
# This code prints Hello, World!
print("Hello, World!")
```

Code Line #22:

```
print("Hello,\nWorld")
```

Code Line #23:

```
Hello,
World
```

Code Line #24:

```
print("Hello, World!\n\nHello, World!")
```

Code Line #25:

```
Hello, World!


Hello, World!
```

fig. 13

THE GROCERY LIST IN MEMORY

	0	1	2	3	4	5	6	7	
8000	E	g	g	s					
8008	M	i	l	k					
8016	W	a	t	e	r				
8024	A	p	p	l	e	s			
8032	S	o	a	p					
8040	C	r	a	c	k	e	r	s	
8048	B	r	e	a	d				
8056	G	i	n	g	e	r			

The grocery list in memory. The vertical column headings are the absolute position in memory, and the numbers in the horizontal top row denote the length of the string.

fig. 14

PYTHON DATA STRUCTURE

STRUCTURE	FORMAT	EXAMPLE
List	`["item1", "item2"]`	`planets = ["Venus", "Earth", "Mars"]`
Tuple	`("item1","item2")`	`flavors = ("grape", "cherry", "lemon")`
Set	`{"item1", "item2"}`	`grades = {"A", "B", "C", "D", "F"}`
Dictionary	`{"key": "value"}`	`names = {"name": "Robert", "age": 42}`

Code Line #26:

```
grocery_list = ["eggs", "milk", "cheese", "pasta"]


planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]


odd_numbers = [1, 3, 5, 7, 9]
```

Code Line #27

```
grocery_list_string = "eggs, milk, cheese, pasta"
```

Code Line #28:

```
random_assortment = ["egg", "tree", 3, "green", 94, "pluto", 3.14]
```

Code Line #29:

```
planets = ("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
```

Code Line #30:

```
print(customers)
```

Code Line #31:

```
{'Sam Sharp', 'Brenda Longmire', 'Veronica March', 'Sylvia Smith', 'Walt Hawkins', 'Steve
Hammersmith', 'Andrea Richards', 'James Smith', 'Vanessa Bush'}
```

Code Line 32:

```
customers = {
        "James Smith",
        "Andrea Richards",
        "Sam Sharp",
        "Brenda Longmire",
        "Veronica March",
        "Sylvia Smith",
        "James Smith",
        "Vanessa Bush",
        "Steve Hammersmith",
        "Brenda Longmire",
        "Sylvia Smith",
        "Steve Hammersmith",
        "Walt Hawkins"
}
```

Code Line #33:

```
print(customer1["name"])
```

Code Line #34:

```
customer3 = {
        "name": "Robert"
        "name": "John"
}
```

Code Line #35:

```
print(customer3)
```

Code Line #36:

```
{'name': 'John'}
```

Code Line #37:

```
walking = False
running = True
```

Code Line #38:

```
# Daily high and low temperature (in Fahrenheit)
temps = [
        [ 66, 34 ],
        [ 57, 25 ],
        [ 49, 45 ]
]
```

Code Line #39:

```
[66, 34]
[57, 25]
[49, 45]
66
34
```

Code Line #40:

```
print(temps[1][2][1])
```

CHAPTER 3
Controlling Program Flow

fig. 15

ESSENTIAL COMPARISON OPERATORS IN PYTHON	
==	Equals
!=	Not equals
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

fig. 16

```
1   a = 1
2   b = 2
3   c = 3
4
5   if a > b:
6      print("a is greater than b")
7      if b!= c:
8      print("but b is not equal to c")
9      else:
10          print("b is equal to c")
11  else:
12     print("a is less than b")
13
```

Visual Studio Code marks improperly formatted code

fig. 17

```
1   a = 1
2   b = 2
3   c = 3
4
5   if a > b:
6       print("a is greater than b")
7       if b ≠ c:
8           print("but b is not equal to c")
9       else:
10          print("b is equal to c")
11  else:
12      print("a is less than b")
13
```

Properly formatted code is not marked. (Note that Visual Studio Code replaces the != that we typed with an equals sign with a slash through it, which better conveys the meaning. We use the != only because we can't make this symbol with our keyboard.)

fig. 18

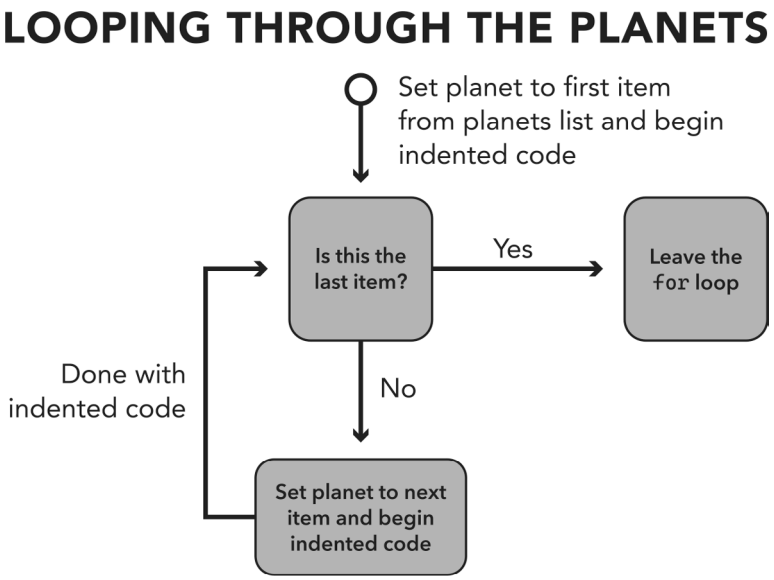
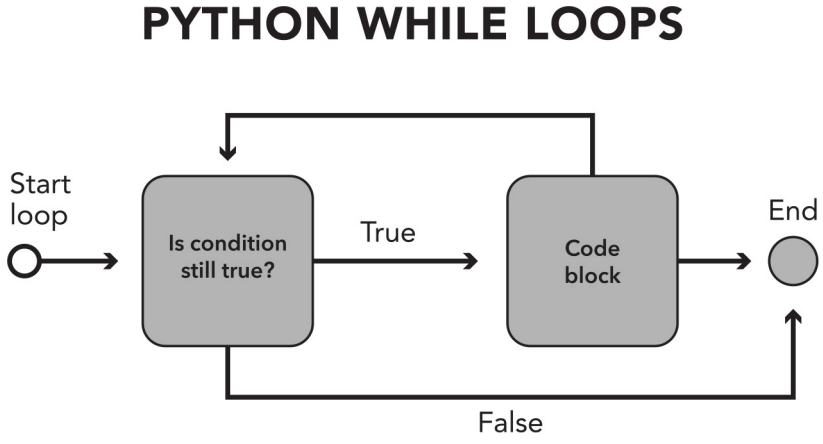


fig. 19



Code Line #41:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

Code Line #42:

```
H
e
l
l
o
,
W
o
r
l
d
!
```

Code Line #43:

```
Planet 0: Mercury
Planet 1: Venus
Planet 2: Earth
Planet 3: Mars
Planet 4: Jupiter
Planet 5: Saturn
Planet 6: Uranus
Planet 7: Neptune
```

Code Line #44:

```
Planet 1: Mercury
Planet 2: Venus
Planet 3: Earth
Planet 4: Mars
Planet 5: Jupiter
Planet 6: Saturn
Planet 7: Uranus
Planet 8: Neptune
```

Code Line #45:

```
0
1
2
3
4
5
6
7
8
9
```

Code Line #46:

```
# While i is less than or equal to 10, display i
i = 1
while i <= 10:
        print(i)
        i += 1
```

Code Line #47:

```
1
2
3
4
5
6
7
8
9
10
```

Code Line #48:

```
while True:
        print("It's true!")
```

Code Line #49:

```
while False:
    print("It's true!")
```

Code Line #50:

```
while True:
    print("Hello, World!")
    break
```

Code Line #51:

```
for i in range(10):
    print(i)
    if i > 5: break
```

Code Line #52:

```
0
1
2
3
4
5
6
```

Code Line #53:

```
for i in range(10):
    if i > 5: break
    print(i)
```

Code Line #54:

```
for i in range(10):
    if i % 2: continue
    print(i)
```

Code Line #55:

```
0
2
4
6
8
```

Code Line #56:

```
for i in range(10):
    for j in range(10):
         print(str(i) + str(j))
```

Code Line #57:

```
from random import seed
from random import randint
```

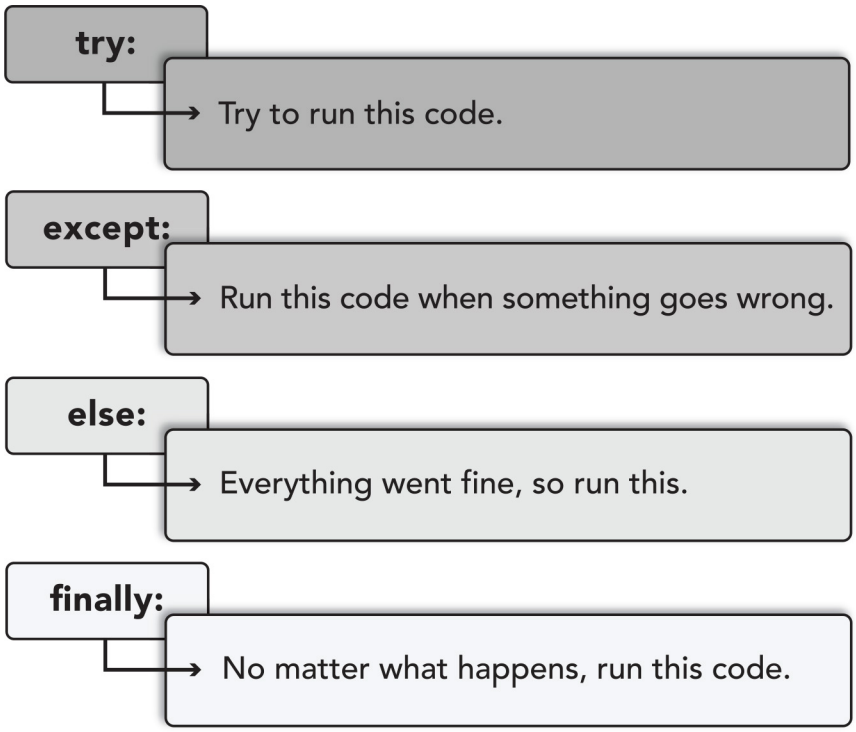
Code Line #58:

```
number = randint(1, 10)
```

Code Line #59:

```
running = True
while running:
        # Do things
```

fig. 20



The Python exception-handling process.

Code Line #60:

```
Traceback (most recent call last):
        File "scratch.py", line 1, in <module>
ZeroDivisionError: division by zero
```

Code Line #61:

```
# Divide a number by zero
a = 7
b = 0

try:
        print(str(a) + " divided by " + str(b) + " is " + str(a / b))
except ZeroDivisionError as e:
        print("Sorry, a problem occurred dividing the numbers.")

print("All done!")
```

Code Line #62:

```
# Divide a number by zero
a = 7
b = 0

try:
        print(str(a) + " divided by " + str(b) + " is " + str(a / b))
except OverflowError as e:
        print("Sorry, a problem occurred dividing the numbers.")

print("All done!")
```

Code Line #63:

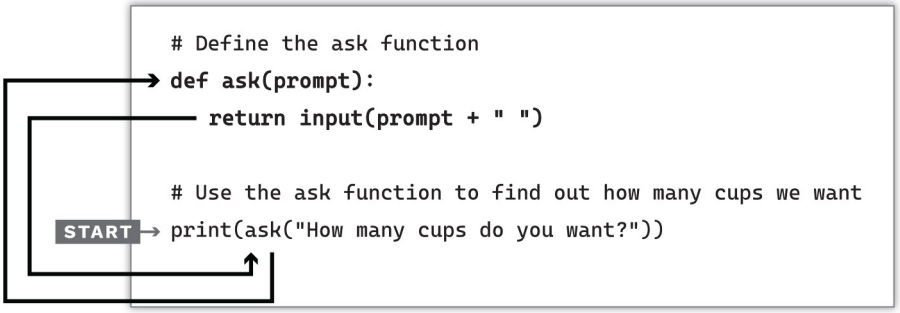
```
except Exception as e:
```

CHAPTER 5
Creating Reusable Tasks with Functions

fig. 21

```
# Define the ask function
def ask(prompt):
   return input(prompt + " ")

# Use the ask function to find out how many cups we want
print(ask("How many cups do you want?"))
```

fig. 22



The flow of program execution in our example.

Code Line #64:

```
# Display the first ten numbers
for i in range(10):
        print(i)
```

Code Line #65:

```
def ask(prompt):
        return input(prompt + " ")
```

Code Line #66:

```
name = ask("What is your name?")
```

Code Line #67:

```
x = 5

def double(x):
        x = x * 2

double(x)
print(x)
```

Code Line #68:

```
x = 5

def double(n):
        return n * 2

x = double(x)
print(x)
```

Code Line #69:

```
def ask(prompt = "Please enter a value: "):
        return input(prompt + " ")
```

Code Line #70:

```
a = ask()
b = ask("What do you want for b?")
```

Code Line #71:

```
def ask(prompt = "Please enter a value: "):
        return input(prompt + " ")
```

Code Line #72:

```
def ask(prompt = "Please enter a value: "):
        if prompt.endswith(" "):
             return input(prompt)
        else:
             return input(prompt + " ")
```

Code Line #73:

```
# This works because the last variable provides a default value
def full_name(first, middle, last, display = False):

# This also works because the last two variables provide a default value
def full_name(first, middle, last = "Last", display = False):

# This doesn't work because a required variable
# comes after one with a default value
def full_name(first = "First", middle, last, display = False):
```

Code Line #74:

```
def full_name(first, middle, last, display = False):
        name = first + " " + middle + " " + last
        if display:
             print(name)
        return name
```

Code Line #75:

```
print(full_name(first = "Robert", middle = "W", last = "Oliver"))
```

Code Line #76:

```
print(full_name(last = "Oliver", first = "Robert", middle = "W"))
```

Code Line #77:

```
# This passes the display value as a keyword argument
print(full_name(last = "Oliver", first = "Robert", middle = "W", display = False))
```

Code Line #78:

```
print(len("Hello, World!"))
```

Code Line #79:

```
def infinity():
        i = 0
        while True:
             yield i
             i += 1
```

Code Line #80:

```
for i in infinity():
        print(i)
```

Code Line #81:

```
bottles = 99
while bottles > 0:
        print(str(bottles) + " bottles of beer on the wall.")
        print(str(bottles) + " bottles of beer.")
        bottles -= 1
        print("Take one down, pass it around,")
        print(str(bottles) + " bottles of beer on the wall.")
print("Take one down, pass it around,")
print(str(bottles) + " bottles of beer on the wall.")
```

CHAPTER 6
Classes

fig. 23

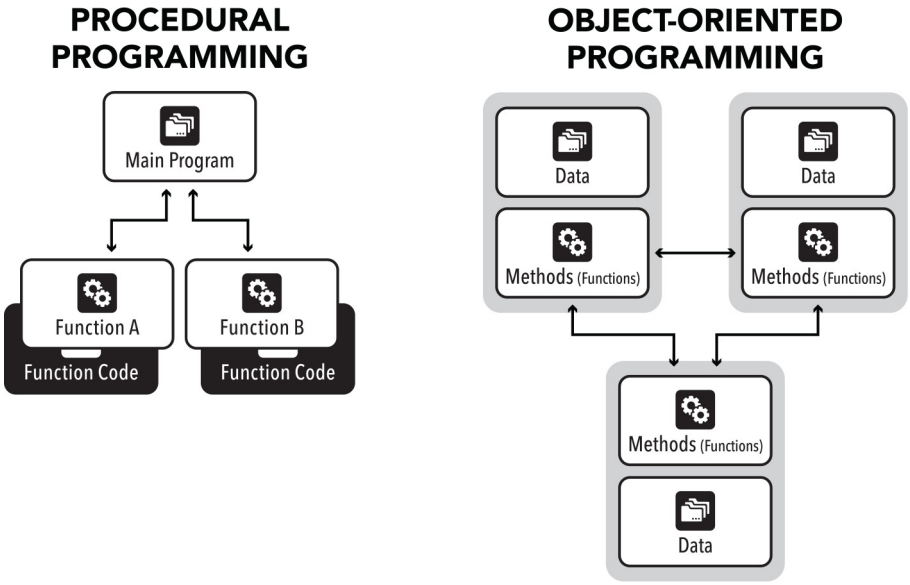


fig. 24

World Class Data

```
# Define the World class
class World:
   # Define our greeting
   greeting =  "Hello, World!"
```

World Class `__init__` Method

```
   # Run this whenever the object is created
   def __init__(self):
        # Print the greeting
        print(self.greeting)
```

Main Program

```
# Use the class World to create a world object named w
w = World()
```

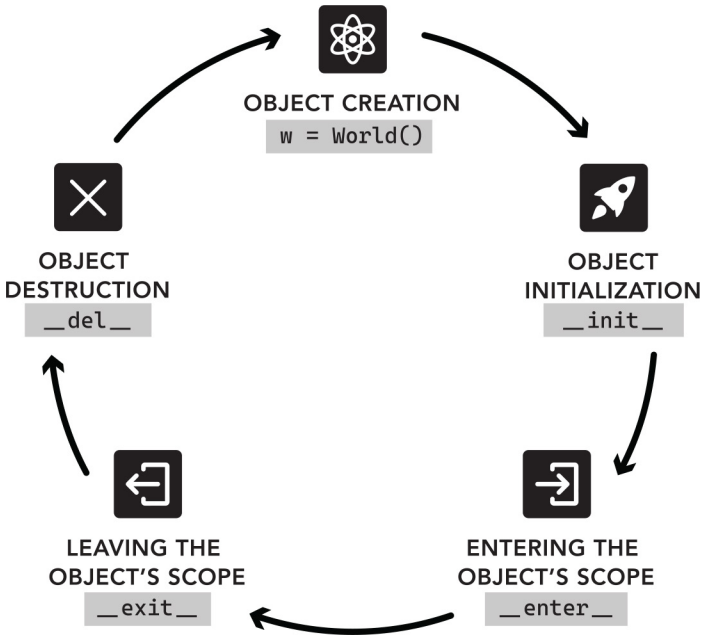
Example code, highlighting the class data, the `__init__` method, and the main program.

fig. 25

```
def __init__ (self, name, city):

c1 = Customer("Sarah", "Atlanta")
```

Comparing the `__init__` method arguments with the creation of the customer object.

fig. 26



Code Line #82:

```
# Define a new class
class Customer:
        def __init__(self):
             name = "Robert"

# Create three objects based on the Customer class
c1 = Customer()
c2 = Customer()
c3 = Customer()
```

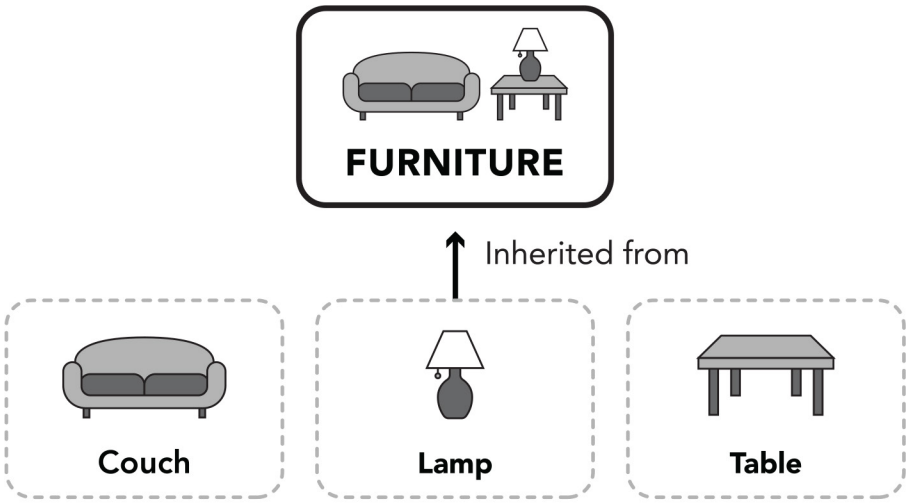
Code Line #83:

```
# Define the World class
class World:
        # Define our greeting
        greeting = "Hello, World!"

# Run this whenever the object is created
def __init__(self):
        # Print the greeting
        print(self.greeting)

# Use the class World to create a world object named w
w = World()
```

Code Line #84

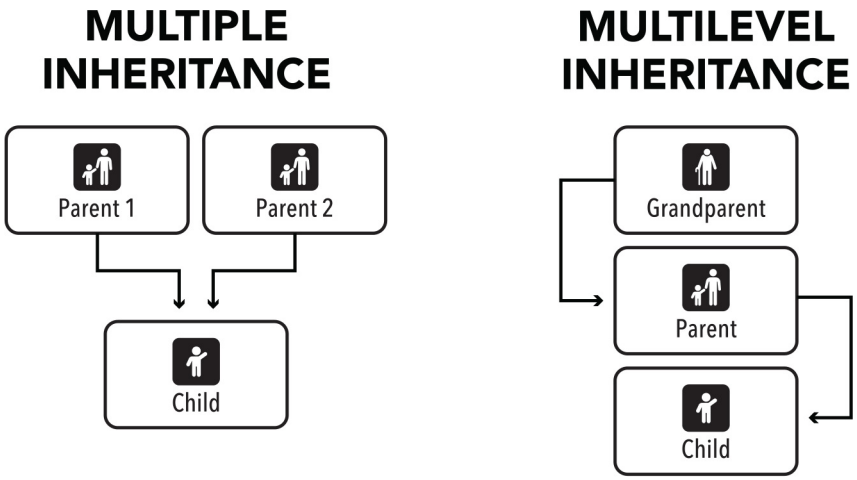
```
class World:
        # Define our greeting
        greeting = "Hello, World!"

        # Run this whenever the object is created
        def __init__(self):
             # Print the greeting
             print(self.greeting)
```

fig. 27



Many classes of furniture, all inherited from the Furniture class.

fig. 28



Multiple inheritance compared to multilevel inheritance.

Code Line #85:

```
# First, let's define the Furniture class
class Furniture:
     def __init__(self, width = 0, height = 0, material = "Wood"):
        self.width = width
        self.height = height
        self.material = material

# Next, let's define the Chair class
class Chair(Furniture):
     def __init__(self, width = 0, height = 0, material = "Wood", arms = True, back = True):
        super().__init__(width, height, material)
        self.arms = arms
        self.back = back
```

Code Line #86:

```
class Furniture:
     def __init__(self, width = 0, height = 0, material = "Wood"):
        self.width = width
        self.height = height
        self.material = material

class Chair(Furniture):
     def __init__(self, material, width = 0, height = 0, arms = True, back = True):
        super().__init__(width, height, material)
        self.arms = arms
        self.back = back

class Bench(Chair):
     pass
```

Code Line #87:

```
{'width': 0, 'height': 0, 'material': 'Metal', 'arms': True, 'back': True}
```

Code Line #88:

```
{'width': 0, 'height': 0, 'material': 'Wood', 'flat': True, 'legs': 4}
```

Code Line #89:

```
# Two quiet, unassuming, regular tables. Nothing fancy here.
# We create them with no parameters and they receive defaults.
a = Table()
b = Table()

# And then there's a weird table named fred. Poor fred.
# He's not like the other tables a and b, he's not flat!
fred = Table(flat = False)
```

Code Line #90:

```
print(bar[2].number)
```

Code Line #91:

```
bar[4].number = 54
```

Code Line #92:

```
print(bar[4].number)
```

Code Line #93:

```
bar.append(Stool)
```

Code Line #94

```
print(len(bar))
```

Code Line #95:

```
class Furniture:
    pass

class Chair(Furniture):
    pass

class Stool(Chair):
    pass

# Create an empty dictionary named bar
bar = {}

# Create several Stool objects
fred = Stool()
marvin = Stool()

# Add them to the bar dictionary
bar["Fred"] = fred
bar["Marvin"] = marvin
```

Code Line #96:

```
bar["Fred"]
```

Code Line #97:

```
# Define a new class
class Customer:
     def __init__(self, name, city):
        self.name = name
        self.city = city
```

Code Line #98:

```
# Include the dataclasses module
from dataclasses import dataclass

# Define a new class
@dataclass
class Customer:
     name: str
     city: str
```

Code Line #99:

```
# Include the dataclasses module
from dataclasses import dataclass

# Define a new class
@dataclass
class Customer:
     name: str
     city: str
     bonus_points: int
```

Code Line #100:

```
@dataclass
class Customer:
     name: str
     city: str
     bonus_points: int = 100
```

Code Line #101:

```
c1 = Customer()
```

Code Line #102:

```
print(c1.bonus_points)
```

Code Line #103:

```
c2 = Customer("John Smith", "Anytown", 200)
print(c2.bonus_points)
```

Code Line #104:

```
@dataclass
class Customer:
     name: str
     city: str
     bonus_points: int = 100
     total_spent: float = 0.00
```

Code Line #105:

```
@dataclass
class Customer:
     name: str
     city: str = "Florence"
     bonus_points: int = 100
     total_spent: float = 0.00
```

Code Line #106:

```
c1 = Customer("Robert")
print(c1)
```

Code Line #107:

```
Customer(name='Robert', city='Florence', bonus_points=100, total_spent=0.0)
```

Code Line #108:

```
@dataclass(init=True, repr=True, eq=True, order=False, frozen=False)
```

Code Line #109:

```
@dataclass
class Customer:
    name: str
    city: str = "Florence"
    bonus_points: int = 100
    total_spent: float = 0.00

c1 = Customer("Robert")
print(repr(c1))
```

Code Line #110:

```
Customer(name='Robert', city='Florence', bonus_points=100, total_spent=0.0)
```

Code Line #111:

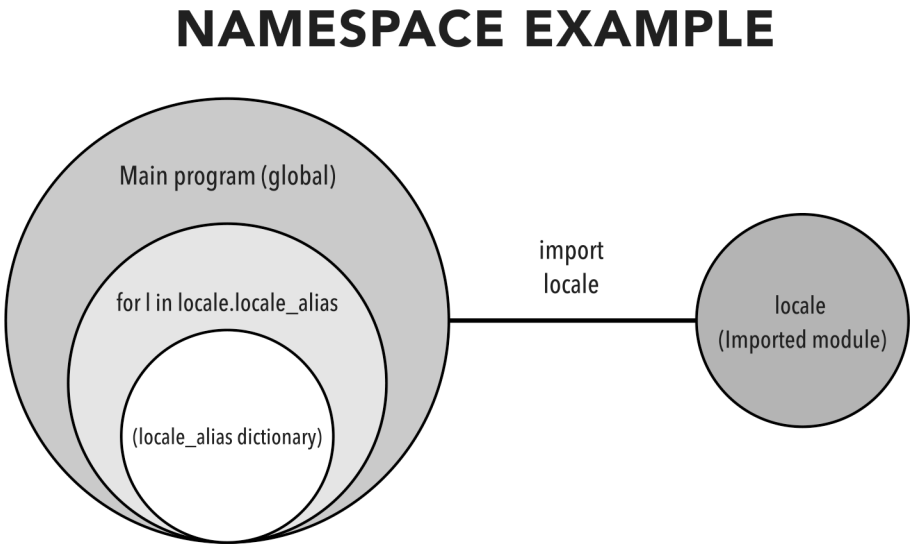
```
c1 = Customer("Robert")
c2 = Customer("Marsha")

print(c1 == c2)
```

Code Line #112:

```
def __eq__(self, other):
    if isinstance(other, Customer):
         return self.name == other.name
    return False
```

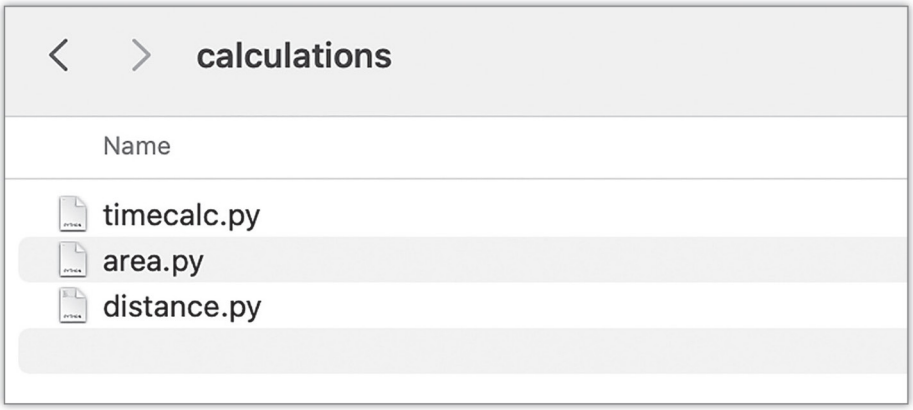
CHAPTER 9
Reusing Code with Modules and Packages

fig. 29



The namespace relationships of the main program, its classes and methods, and the imported module named `locale`.

fig. 30



The `time`, `area`, and `distance` modules (i.e., Python code files) in a folder named `calculations`, as shown in the macOS Finder file browser.

Code Line #113:

```
import locale
for l in locale.locale_alias:
     print(l)
```

Code Line #114:

```
import locale
```

Code Line #115:

```
import locale as robert
```

Code Line #116:

```
from email import parser
```

Code Line #117:

```
vars(email)
```

Code Line #118:

```
dir(email)
```

Code Line #119:

```
help(email)
```

Code Line #120:

```
3 kilometers is 1.8645121193287757 miles.
3.0 miles is 4.827 kilometers.
```

Code Line #121:

https://docs.python.org/3/py-modindex.html

Code Line #122:

```
class Area:
     pass
```

Code Line #123:

```
class Timecalc:
     pass
```

Code Line #124:

```
import area
import distance
import timecalc
```

Code Line #125:

```
from .distance import *
from .area import *
from .timecalc import *
```

Code Line #126:

```
from .distance import Distance
```

Code Line #127:

```
d = calculations.Distance(5)
```

CHAPTER 10
Advanced Strings

fig. 31

The quick brown fox jumps over the lazy dog.

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog.']
```

Splitting strings into lists in Python.

fig. 32

SYMBOL	OBJECTIVE
\b	Matches empty string at the beginning or end of a word (i.e., a word boundary).
\w+	\w matches any single word character (i.e., a-z, A-Z, 0-9). The + causes the \w to match unlimited times.
(\w+)	The parentheses around the \w+ tell Python to capture (save for later) what is matched inside them.
\s+	\s matches any single whitespace character (including but not limited to spaces). Unlike \b, this doesn't have to be at the beginning or end of a word. The + causes the \s to match unlimited times.
\1	Matches the first capturing group again, which is (\w+). The 1 denotes the first group.
\b	Matches empty string at the beginning or end of a word.

Regular expression metacharacters and symbols in the regex `"\b(\w+)\s+\1\b"` as shown in the code the example.

fig. 33

FREQUENTLY USED FORMATTING CODES			
CODE	DESCRIPTION	EXAMPLE	DISPLAY
:n	integer format	`"There are {:n} continents."`	There are 7 continents.
:f	Floating point number	`"Your total is ${:.2f}."`	Your total is $6.95.
:,	Use comma-separators	`"In 2020, the USA had {:,} people"`	In 2020, the USA had 329,500,000 people.
:%	Percentage format	`"Take an additional {.0%} off today!"` *(the 0 defines decimal places in percentage display)*	Take an additional 20% off today!
:e	Exponent (scientific) notion	`"The speed of light is {:e} m/s."`	The speed of light is 3.000000e+08 m/s.
:E	Exponent (scientific) notion (Capital E)	`"The speed of light is {:E} m/s."`	The speed of light is 3.000000E+08 m/s.

Code Line #128:

```
Hello, World!
… was replaced with …
Hello, Reader!
```

Code Line #129:

```
a = a.replace("World", "Reader")
print(a)
```

Code Line #130:

```
PYTHON QUICKSTART GUIDE
python quickstart guide
```

Code Line #131:

```
There are 7 letter s in:
She sells seashells by the seashore.
```

Code Line #132:

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog.']
```

Code Line #133:

```
['123', '45', '6789']
```

Code Line #134:

```
['delimiter', 'module', 'package', 'class', 'object']
```

Code Line #135:

```
Hello is in the string.
```

Code Line #136:

```
hello is in the string.
```

Code Line #137:

```
match.span()
```

Code Line #138:

```
(10, 14)
```

Code Line #139:

```
Old text: The quick gray fox jumped over the lazy dog!
New text: The quick grey fox jumped over the lazy dog!
```

Code Line #140:

```
text = """This is a multiline string.
It has multiple lines to it.
So, fittingly, it's called a multiline string.
When you type a string and hit ENTER,
a special character is inserted in the string.
That special character is a backslash followed by n."""
```

Code Line #141:

```
re.search("\AThis", text)
```

Code Line #142:

```
re.search says it has an e in it.
```

Code Line #143:

```
['The', 'quick', 'brown', 'fox', 'is', 'fast!']
['The', 'quick', 'brown', 'fox', 'is', 'fast', '']
```

Code Line #144:

```
The+quick+brown+fox+is+fast!
```

Code Line #145

```
"Hello, {}!"
```

Code Line #146

```
total = 6.95
message = "Your total is ${:.2f}."


print(message.format(total))
```

Code Line #147:

```
Your total is $6.95.
```

Code Line #148:

```
data = """To compress data, we'll need a long string.
Not a short string. No, that would be too small.
To get any meaningful benefit from compression,
you must use a decent length of data or else the
overhead of compression isn't worth the gains.
This will be enough data, containing enough redundant
patterns, to be compressible."""
```

Code Line #149:

```
import zlib
```

Code Line #150:

```
encoded_data = data.encode()
```

Code Line #151:

```
# Load the zlib module
import zlib
# Define our data
data = """To compress data, we'll need a long string.
Not a short string. No, that would be too small.
To get any meaningful benefit from compression,
you must use a decent length of data or else the
overhead of compression isn't worth the gains.
This will be enough data, containing enough redundant
patterns, to be compressible."""

# Compress the data
compressed_data = zlib.compress(data.encode())

# Display stats
data_len = len(data)
compressed_data_len = len(compressed_data)
print("Length of uncompressed data: " + str(data_len))
print("Length of compressed data: " + str(compressed_data_len))
```

CHAPTER 11
Math in Python

fig. 34

OPERATION	SYMBOL	EXAMPLE
Addition	+	3 + 5 + 8
Subtraction	–	5 - 2 = 3
Multiplication	*	5 * 5 = 25
Division	/	6 / 3 = 2
Exponent	**	3 ** 3 = 27

The basic symbols involved in Python math.

fig. 35

ORDER OF OPERATIONS					
P (Parentheses)	**E** (Exponent)	**M** (Multiply)	**D** (Divide)	**A** (Add)	**S** (Subtract)
B (Brackets)	**O** (Order)	**D** (Divide)	**M** (Multiply)	**A** (Add)	**S** (Subtract)
( )	**$\sqrt{x}$ or x^2**	**÷ or x**		**+ or –**	

The order of operations in Python.

fig. 36

COMMON MATH STANDARD MODULE FUNCTIONS		
FUNCTION	DESCRIPTION	EXAMPLE
floor(n)	Find largest integer less than or equal to n	`print(str(math.floor(5.9)))` `Result: 5`
ceil(n)	Find smallest integer greater or equal to n	`print(str(math.ceil(4.9)))` `Result: 5`
fabs(n)	Return the absolute value of n as float	`print(str(math.fabs(-5)))` `Result: 5.0`
fmod(x, y)	Return remainder from x / y	`print(str(math.fmod(3, 2)))` `Result: 1.0`
cos(n)	Find cosine of n	`print(str(math.cos(1)))` `Result: 0.5403023058681398`
sin(n)	Find sine of n	`print(str(math.sin(1)))` `Result: 0.8414709848078965`
tan(n)	Find tangent of n	`print(str(math.tan(1)))` `Result: 1.5574077246549023`
pi	Return the value of pi	`print(math.pi))` `Result: 3.141592653589793`

fig. 37

CODE	DESCRIPTION	RESULTS
`%a`	Abbreviated weekday	Sun, Mon, Tue, etc.
`%A`	Weekday full name	Sunday, Monday, etc.
`%w`	Weekday as number (i.e., day of the week)	Sunday is 0, Saturday is 6
`%d`	Day of the month	01 through 31
`%b`	Abbreviated month	Jan, Feb, Mar, etc.
`%B`	Month full name	January, February, etc.
`%m`	Month number	01 through 12
`%y`	2-digit year	00 through 99
`%Y`	4-digit year	0001 through 9999
`%H`	Hour (24-hour clock)	00 through 23
`%I`	Hour (12-hour clock)	00 through 12
`%p`	AM or PM	AM, PM
`%M`	Minute	00 through 59
`%S`	Second	00 through 59
`%f`	Microsecond	000000 through 999999
`%z`	UTC offset	(empty), +0000, -0100, etc.
`%Z`	Time zone	(empty), UTC, GMT, etc.
`%j`	Day of the year	001 through 366 (366 is possible in leap year)
`%U`	Week number of the year	00 through 53
`%W`	Week (Mon as first day) number of the year	00 through 53
`%c`	Locale's date and time representation	Mon Apr 4 21:31:00 2022
`%x`	Locale's date representation	04/04/2022
`%X`	Locale's time representation	21:31:00
`%%`	A literal '%' character	%

Possible `strftime` format codes. These values are adjusted to meet the standards of the locale selected on the computer. For example, if the selected country typically lists year first, `%c` and `%x` will reflect this.

Code Line #152:

```
# PEMDAS
result = (5 * 4) / 6 - (1 + 2) + 3 ** 2
print(result)
```

Code Line #153:

```
9.333333333333334
```

Code Line #154:

https://docs.python.org/3/library/math.html

Code Line #156:

```
5
5.0
```

Code Line #157:

```
1
1
2
```

Code Line #158:

```
n = 0.2 + 0.2 + 0.2
```

Code Line #159:

```
# Display n with 30 decimal precision
print("The total is {:.30f}".format(n))
```

Code Line #160:

```
0.600000000000000088817841970013
```

Code Line #161:

```
0.34 34.0
```

Code Line #162:

```
print(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Code Line #163:

```
1 2 3 4 5 6 7 8 9
```

Code Line #164:

```
print(percent(0.42))
```

Code Line #165:

https://docs.python.org/3/library/statistics.html

Code Line #166:

```
# Import datetime
import datetime

# Get the current time and date
now = datetime.datetime.now()

# Display the year
print(now.year)

# Display the month
print(now.month)
```

Code Line #167:

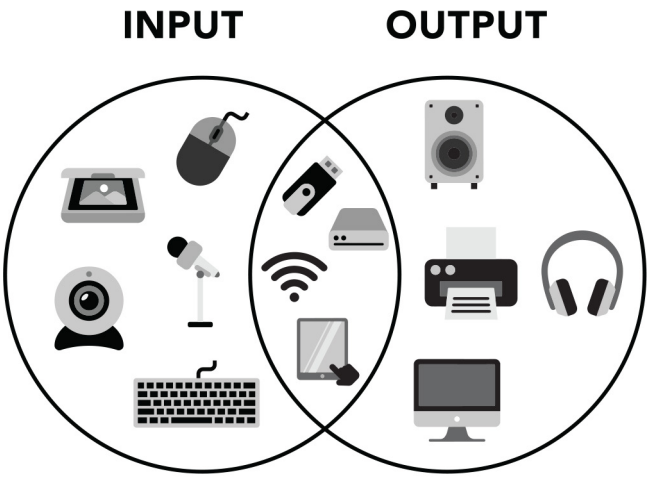
```
future = datetime.datetime(2023, 12, 25, 0, 0)
```

Code Line #168:

```
pip3 install numpy
```

CHAPTER 12
Input and Output

fig. 38



A wide variety of input and output devices. Some serve both roles.

Code Line #169:

```
Original  : The lazy red fox slept instead of jumping over a dog.
From disk : The lazy red fox slept instead of jumping over a dog.
```

Code Line #170:

```
Original  : The lazy red fox slept instead of jumping over a dog.
From disk : The lazy red fox
```

Code Line #171:

```
with open("fox.txt", mode = "rb") as f:
    while (c := f.read(32)):
         # Do something with data in c
```

Code Line #172:

```
with open("fox.txt", mode = "r") as f:
    while (l := f.readline()):
         # Do something with the line in l
```

Code Line #173:

```
Original  : The lazy red fox slept instead of jumping over a dog.
The lazy dog ignored the fox and slept as well.
From disk : ['The lazy red fox slept instead of jumping over a dog.\n', 'The lazy dog ignored the
fox and slept as well.']
```

Code Line #174:

```
Testing, 123!
Testing, 123!
Wrote 14 bytes to stdout.
```

Code Line #175:

```
python3 inout.py > inout.txt
```

Code Line #176:

```
python inout.py > inout.txt
```

Code Line #177:

```
Original data : ['Jim Smith', 'Amber Dobson', 'Al James']
Loaded data   : ['Jim Smith', 'Amber Dobson', 'Al James']
```

Code Line #178:

```
# Add this to the top of the code
import ssl

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

# Then change: response = urlopen(url)
# to the following code:

urlopen(url, context=ctx)
```

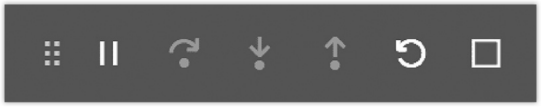
Code Line #179
www.mediawiki.org/wiki/API:Main_page

CHAPTER 14
Debugging Python Code

fig. 39

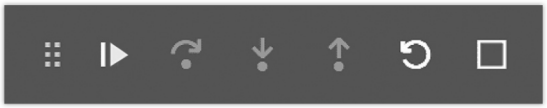
LEVEL	PRIORITY	DESCRIPTION
`logging.DEBUG`	10	Provides the most detail. Useful for diving deep into problems.
`logging.INFO`	20	Used for basic information about execution.
`logging.WARNING`	30	Used for important warnings about potential hazards.
`logging.ERROR`	40	Used for events that could stop the program.
`logging.CRITICAL`	50	Reserved for the most serious problems.

fig. 40



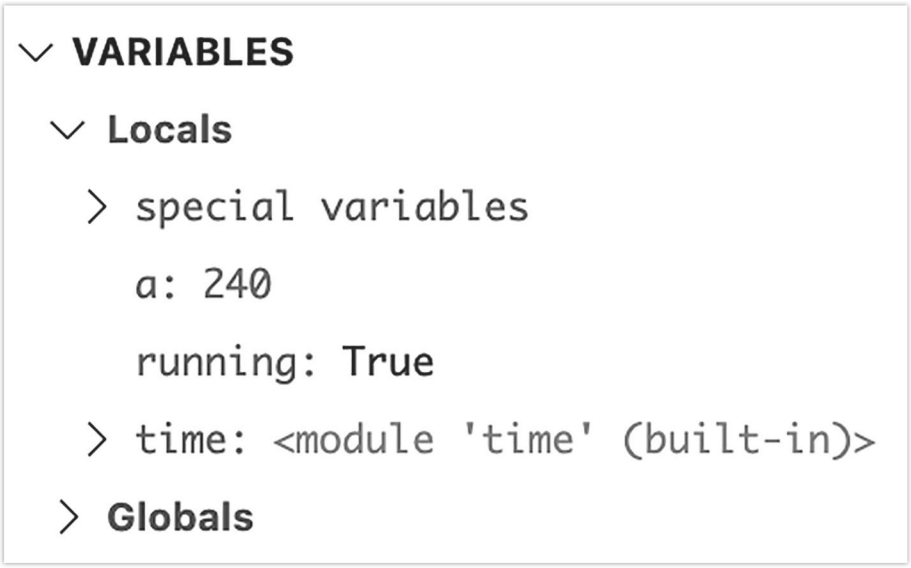
The debug toolbar bar that appears whenever a program is run in Visual Studio Code. The *pause* button is the first icon on the left (the two vertical lines).

fig. 41



The first icon resumes execution. The second steps over the current line. The third steps into the next function call or loop and pauses at the first line. The fourth steps out of the current function by finishing it, then pausing at the line of code that called it. The next icon restarts execution, and the square icon (stop) ends all execution.

fig. 42



The variables pane of our sample debugging program.

fig. 43

```
6  while True:
7      print("Hello, World!")
8      print("a is " + str(a))
9      a += 1
10     time.sleep(1)
11
```

A breakpoint has been set on line 8.

Code Line #180:

```
print("Reached first function call.")
```

Code Line #181:

```
print("Inside for loop.")
```

Code Line #182:

```
# Log a DEBUG message
logging.debug("This is a debug message.")

# Log an INFO message
logging.info("This is an info message.")

# Log a WARNING message
logging.warning("This is a warning message.")

# Log an ERROR message
logging.error("This is an error.")

# Log a CRITICAL message
logging.critical("Something major went wrong!")
```

Code Line #183:

```
WARNING:root:This is a warning message.
ERROR:root:This is an error.
CRITICAL:root:Something major went wrong!
```

Code Line #184:

```
web_logger = logging.getLogger("jobs")
```

Code Line #185:

```
2022-04-21 00:45:43,639 - Something REALLY bad happened!
```

Code Line #186:

```
Hello, World!
a is 0
Hello, World!
a is 1
```

fig. 44

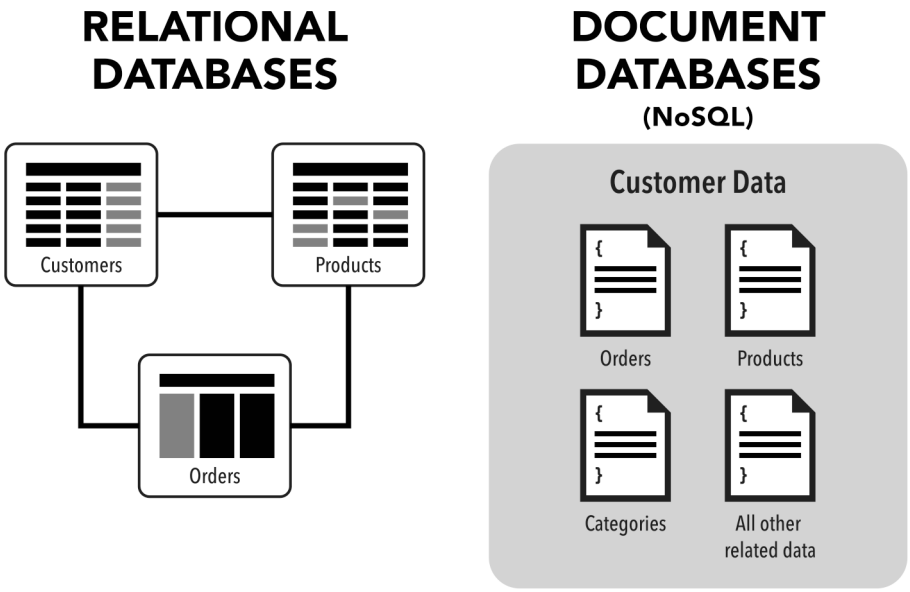
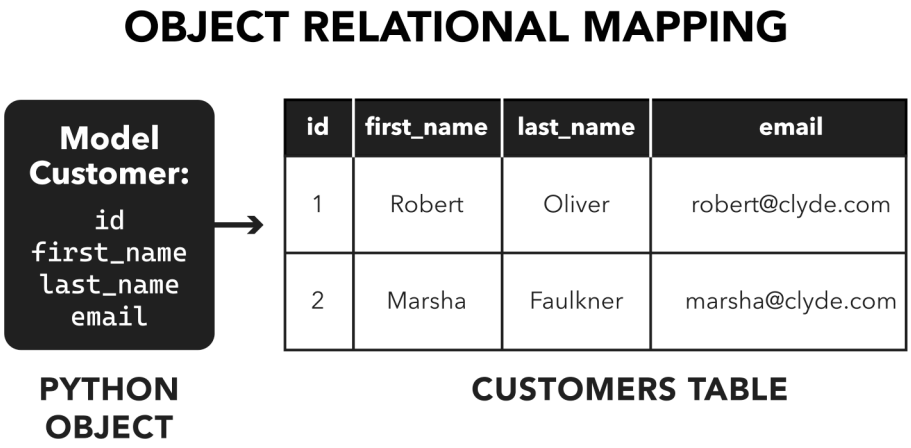

RELATIONAL DATABASES
Customers
Products
Orders
DOCUMENT DATABASES
(NoSQL)
Customer Data
Orders
Products
Categories
All other related data

fig. 45


OBJECT RELATIONAL MAPPING
Model Customer:
id
first_name
last_name
email
PYTHON OBJECT
id
first_name
last_name
email
1
Robert
Oliver
robert@clyde.com
2
Marsha
Faulkner
marsha@clyde.com
CUSTOMERS TABLE

Code Line #187:

```
pip3 install web.py
```

Code Line #188:

```
http://0.0.0.0:8080
```

Code Line #189:

http://127.0.0.1:8080/

Code Line #190:

```
http://0.0.0.0:8080/You
```

Code Line #191:

```
pip3 install flask
```

Code Line #192:

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
       {
            "name": "Python: Flask",
            "type": "python",
            "request": "launch",
            "module": "flask",
            "env": {
               "FLASK_APP": "hello-flask.py",
               "FLASK_ENV": "development"
            },
            "args": [
               "run",
               "--no-debugger"
            ],
            "jinja": true,
            "justMyCode": true
       }
    ]
}
```

Code Line #193:

```
* Serving Flask app 'hello-flask.py' (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Restarting with stat
```

Code Line #194:

```
pip3 install mysql-connector-python
```

Code Line #195:

https://dev.mysql.com/doc/mysql-startstop-excerpt/8.0/en

Code Line #196:

```
coffeeshop/
     manage.py
     coffeeshop/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Code Line #197:

```
from django.http import HttpResponse

def index(request):
     return HttpResponse("Hello, World!")
```

Code Line #198:

```
from django.urls import path

from . import views

urlpatterns = [
     path('', views.index, name='index'),
]
```

Code Line #199:

```
from django.db import models

class Customer(models.Model):
     first_name = models.CharField(max_length=64)
     last_name = models.CharField(max_length=64)
```

Code Line #200:
```
C:\Users\You\Source\sqlite3.exe
```

Code Line #201:
```
%USERPROFILE%\Source\sqlite3.exe
```

Code Line #202:
```
# Debian, Ubuntu, Linux Mint, and other Debian-based distros
apt install sqlite3

# Fedora, Red Hat Enterprise Linux, and related distros
yum install sqlite

# Arch, Manjaro, and other Arch-based distros
pacman -S sqlite
```

Code Line #203:
```
sqlite3
```

Code Line #204:
```
%USERPROFILE%\Source\sqlite3.exe
```

Code Line #205:
```
sqlite3 sTunes.db
```

Code Line #206:
```
sqlite3 sTunes.db
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite>
```

Code Line #207:
```
running = True
while running:
    cmd = input("sqlite> ")
    # Use regular expressions to parse cmd
    # Check for matches against defined commands
    if cmd == ".quit":
         running = False
```

Code Line #208:
```
.schema tracks
```

Code Line #209:

```
CREATE TABLE IF NOT EXISTS "tracks"
(
     [TrackId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
     [Name] NVARCHAR(200)  NOT NULL,
     [AlbumId] INTEGER,
     [MediaTypeId] INTEGER  NOT NULL,
     [GenreId] INTEGER,
     [Composer] NVARCHAR(220),
     [Milliseconds] INTEGER  NOT NULL,
     [Bytes] INTEGER,
     [UnitPrice] NUMERIC(10,2)  NOT NULL,
     FOREIGN KEY ([AlbumId]) REFERENCES "albums" ([AlbumId])
        ON DELETE NO ACTION ON UPDATE NO ACTION,
     FOREIGN KEY ([GenreId]) REFERENCES "genres" ([GenreId])
        ON DELETE NO ACTION ON UPDATE NO ACTION,
     FOREIGN KEY ([MediaTypeId]) REFERENCES "media_types" ([MediaTypeId])
        ON DELETE NO ACTION ON UPDATE NO ACTION
     );
CREATE INDEX [IFK_TrackAlbumId] ON "tracks" ([AlbumId]);
CREATE INDEX [IFK_TrackGenreId] ON "tracks" ([GenreId]);
CREATE INDEX [IFK_TrackMediaTypeId] ON "tracks" ([MediaTypeId]);
```

Code Line #210:

```
select * from tracks;
```

Code Line #211:

```
select * from albums limit 5;


1|For Those About To Rock We Salute You|1
2|Balls to the Wall|2
3|Restless and Wild|2
4|Let There Be Rock|1
5|Big Ones|3
```

Code Line #212:

```
.mode table
```

Code Line #213:

```
+-----------+-----------------------------------------+------------+
|AlbumId    |                  Title                  |  ArtistId  |
+-----------+-----------------------------------------+------------+
| 1         |  For Those About To Rock We Salute You  | 1          |
| 2         | Balls to the Wall                       | 2          |
| 3         | Restless and Wild                       | 2          |
| 4         | Let There Be Rock                       | 1          |
| 5         | Big Ones                                | 3          |
+-----------+-----------------------------------------+------------+
```

Code Line #214:

```
insert into table (column1, column2, column3) values (X, Y, Z);
```

Code Line #215:

```
update table set column1 = value1, column2 = value2 where id = X;
```

Code Line #216:

```
delete from table where id = X;
```

Code Line #217:

```
# Don't run this unless you really want
# to see how a delete query works

query = "delete from tracks where TrackId = 3500"
db.execute(query)
```

Code Line #218:

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s
OK
```

Code Line #219:

```
def say_greeting(name = "World"):
     return("Well hello, " + name + "!")
```

Code Line #220:

```
FF
======================================================================
FAIL: test_greeting_with_name (__main__.GreetingTests)
----------------------------------------------------------------------
Traceback (most recent call last):
     File "/Users/rwoliver2/Source/python-book/greeting_tests.py", line 13, ↦
     ↪ in test_greeting_with_name
        self.assertEqual(greeting.say_greeting("Robert"), "Hello, Robert!")
AssertionError: 'Well hello, Robert!' != 'Hello, Robert!'
- Well hello, Robert!
? ^^^^^^
+ Hello, Robert!
? ^



======================================================================
FAIL: test_greeting_without_name (__main__.GreetingTests)
----------------------------------------------------------------------
Traceback (most recent call last):
     File "/Users/rwoliver2/Source/python-book/greeting_tests.py", line 10, ↦
     ↪ in test_greeting_without_name
        self.assertEqual(greeting.say_greeting(), "Hello, World!")
AssertionError: 'Well hello, World!' != 'Hello, World!'
- Well hello, World!
? ^^^^^^
+ Hello, World!
? ^



----------------------------------------------------------------------
Ran 2 tests in 0.000s
FAILED (failures=2)
```

Code Line #221:

```
AssertionError: 'Well hello, World!' != 'Hello, World!'
```

Code Line #222:

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

Code Line #223:

```
self.assertEqual(greeting.say_greeting(), "Hello, World!")
self.assertEqual(greeting.say_greeting("Robert"), "Hello, Robert!")
```

Code Line #224:

```
# If greeting.say_greeting() returns "Hello, World!", it passes.
    self.assertEqual(greeting.say_greeting(), "Hello, World!")
```

Code Line #225:

```
# If greeting.say_greeting() returns "Hello, World!", it fails.
    self.assertNotEqual(greeting.say_greeting(), "Hello, World!")
```

Code Line #226:

```
# These assertions pass
self.assertTrue(True)
self.assertTrue(1 == 1)

    # These fail
self.assertTrue(False)
self.assertTrue(1 == 2)

    # This passes because it will return True
self.assertTrue(greeting.say_greeting())
```

Code Line #227:

```
# These pass
self.assertFalse(False)
self.assertFalse(1 == 2)

    # These assertions fail
self.assertFalse(True)
self.assertFalse(1 == 1)

    # This fails because it will return True
self.assertTrue(greeting.say_greeting())
```

Code Line #228:

```
def write_file(data, filename):
     # Write the file
     return True
```

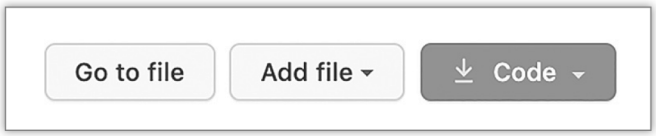
Code Line #229:

```
def write_file(data, filename):
     try:
        # Write the file
     except Exception as e:
        # Handle the exception, then return False
        return False
     # If we made it this far, it was successful, return True
     return True
```

Code Line #230:

```
self.assertTrue(write_file("Hello", "test.txt"))
```

fig. 46



Code Line #231:

```
cd ~/Source
```

Code Line #232:

```
cd %HOMEPATH%\Source
```

Code Line #233:

```
git clone https://github.com/rwoliver2/Python-CoffeeShopSimulator.git
```

Code Line #234:

```
git add filename.py
```

Code Line #235:

```
git add .
```

Code Line #236:

```
git status
```

Code Line #237:

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Code Line #238:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
     (use "git add <file>..." to update what will be committed)
     (use "git restore <file>..." to discard changes in working directory)
       modified:   main.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Code Line #239:

```
git commit -m "A note about your changes."
```

Code Line #240:

```
git commit -a -m "A note about your changes."
```

Code Line #241:

```
git push origin main
```

Code Line #242:

```
git checkout -b branch_name
```

Code Line #243:

```
git push origin branch_name
```

Code Line #244:

```
git checkout branch_name
```

Code Line #245:

```
git diff
```

Code Line #246:

```
diff --git a/main.py b/main.py
index df53dae..41f762b 100644
--- a/main.py
+++ b/main.py
@@ -1,5 +1,5 @@
 # ClydeBank Coffee Shop Simulator 4000
-# Copyright (C) 2023 ClydeBank Media, All Rights Reserved.
+# Copyright (C) 2024 ClydeBank Media, All Rights Reserved.
```

Code Line #247:

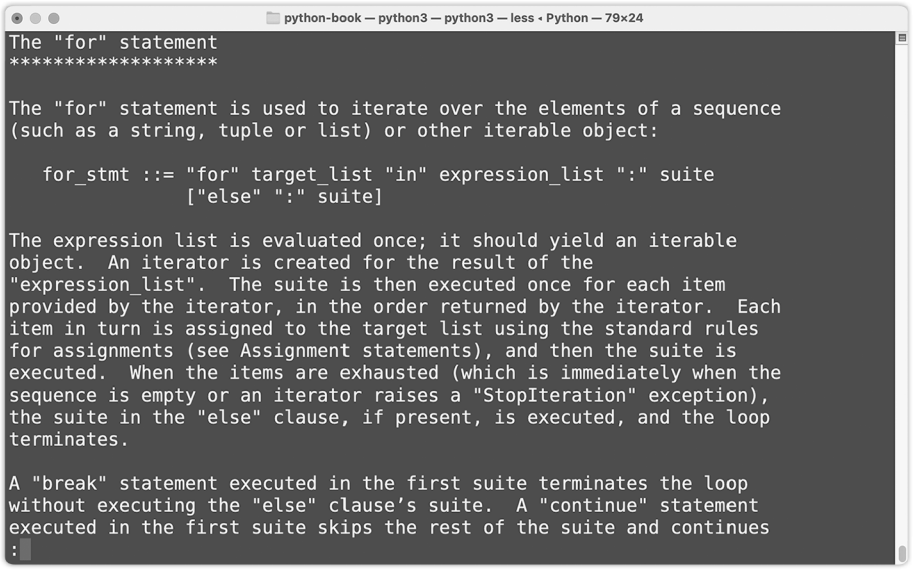
```
git log
```

Code Line #248:

```
commit 6dd258b5273bc4a13eee201fd6304efc61baef5a (HEAD -> main, origin/main, origin/HEAD)
Author: Robert W. Oliver II <118407+rwoliver2@users.noreply.github.com>
Date:   Sun Jul 3 23:11:07 2022 -0500

    Update README.md

commit a2af786f19023bd2ac2d023ce8b2f2ed664f733b
Author: Robert W. Oliver II <118407+rwoliver2@users.noreply.github.com>
Date:   Sun Jul 3 23:07:07 2022 -0500

    Initial commit.
```

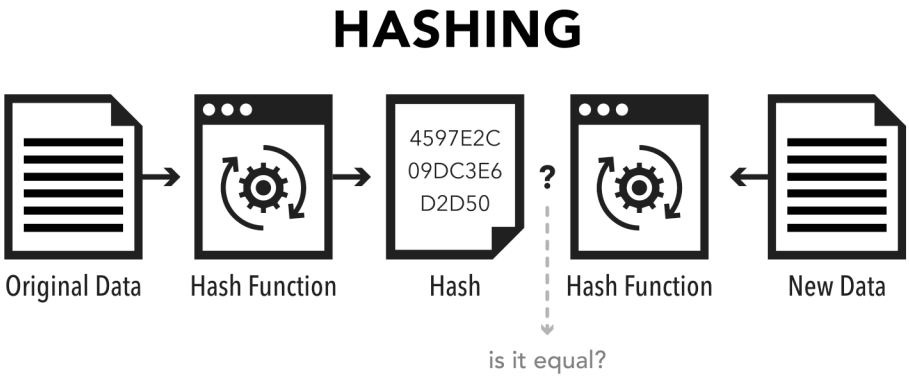
CHAPTER 19
The Junk Drawer

fig. 47



The help documentation for the `for` statement.

fig. 48



Hashing can validate the contents of two sets of data. The hash function is run on both sets of data (the original and the new) and if the hashes match, the data are equal.

fig. 49

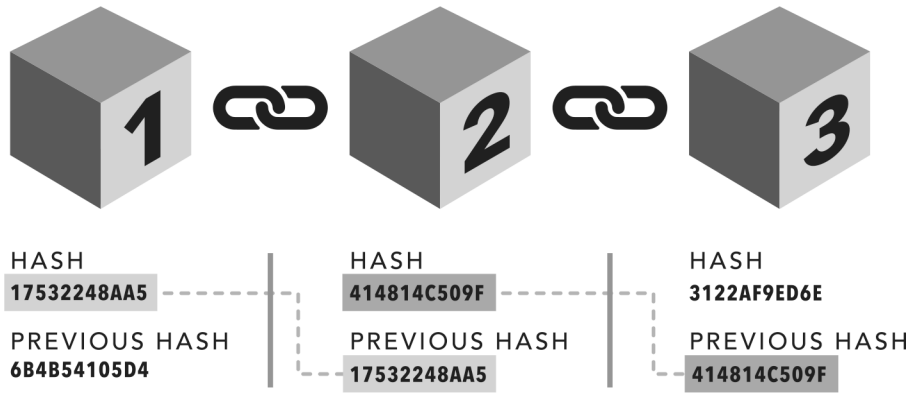


fig. 50

	A	B	C	D	E
1	Month	Coffee	Tea	Hot Chocolate	Espresso
2	Jan	523	301	507	332
3	Feb	621	339	501	339
4	Mar	512	218	497	401
5	Apr	511	401	324	385
6					

A simple sales report spreadsheet with delicious beverages.

fig. 51

```
class Distance:
    def __init__(self, km):
        self._km = km
    @property
    def km(self):
        return self._km
    @km.setter
    def km(self, value):
        self._km = value
    @property
    def miles(self):
        return self._km / 1.609
    @miles.setter
    def miles(self, value):
        self._km = value * 1.609

distance2 = Distance(3)
print("3 kilometers is " + str(distance2.miles) + " miles.")
distance2.miles = 3
print(str(distance2.miles) + " miles is " + str(distance2.km) + " kilometers.")
```

The original `km-miles.py` file.

fig. 52

```
a
??+b?@sVGdd?d?Zed?Zedeej?d?de_eeej?deej?d?dSc@sHeZdZdd?Zedd??Zejdd??Zedd??Zejd??Zd      S)
DistancecCs
||_dS?N?Z_km)?self?km?r?
                        km-miles.py__init__szDistance.__init__cCs|jSrr?rrrrsz
                                                                                    Distance.kmcCs
||_dSrr?r?valuerrrrscCs
|jd?NgX9??v???rr       rrr?miles
szDistance.milescCs|d|_dSr
                            rr
sN__name__?
__module__?
rrrrrs     __qualname__propertyr?setterr


r?z3 kilometers is z miles.z
 miles is z _
```

The bytecode `km-miles.pyc` file.

Code Line #249:

```
help()
```

Code Line #250:

```
# Create a shopping list
shopping_list = ["Eggs", "Butter", "Milk", "Sausage", "Apples"]

# Sort it
shopping_list.sort()

# Display the results
print(shopping_list)
```

Code Line #251:

```
['Apples', 'Butter', 'Eggs', 'Milk', 'Sausage']
```

Code Line #252:

```
# Create a shopping list
shopping_list = ["Eggs", "Butter", "Milk", "Sausage", "Apples"]

# Sort it in reverse
shopping_list.sort(reverse = True)

# Display the results
print(shopping_list)
```

Code Line #253:

```
['Sausage', 'Milk', 'Eggs', 'Butter', 'Apples']
```

Code Line #254:

```
[{'name': 'Coffee', 'price': 3.5},
 {'name': 'Hot Chocolate', 'price': 3.99},
 {'name': 'Orange Juice', 'price': 1.99},
 {'name': 'Soda', 'price': 1.75},
 {'name': 'Tea', 'price': 2.99}]
```

Code Line #255:

```
VAR="Hello" python3 myprogram.py
```

Code Line #256:

```
os.environ["VAR_NAME"]
```

Code Line #257:

```
os.environ["VAR_NAME"] = "Hello"
```

Code Line #258:

```
python3 yourprogram.py
```

Code Line #259:

```
/usr/bin/env /opt/homebrew/bin/python3
/Users/rwoliver2/.vscode/extensions/ms-python.python-
2022.4.1/pythonFiles/lib/python/debugpy/launcher 61829 --
/Users/rwoliver2/Source/python-book/yourprogram.py
```

Code Line #260:

```
python3 yourprogram.py help
```

Code Line #261:

```
python3 yourprogram.py infile.csv outfile.csv
```

Code Line #262:

```
python3 cmdlinetest.py infile.csv outfile.csv
```

Code Line #263:

```
Input file: infile.csv
Output file: outfile.csv
```

Code Line #264:

```
# Import the sys module
import sys

input_file = sys.argv[1]
output_file = sys.argv[2]

print("Input file: " + input_file)
print("Output file: " + output_file)
print("Zero index position: " + sys.argv[0])
```

Code Line #265:

```
python3 infile.csv outfile.csv
```

Code Line #266:

```
Input file: infile.csv
Output file: outfile.csv
Zero index position: cmdlinetest.py
```

Code Line #267:

```
Input file: cmdlinetest.py
Output file: infile.csv
```

Code Line #268:

```
# A very simple lambda
greet = lambda name: "Hello, " + name + "!"
```

Code Line #269:

```
greet("Robert")
```

Code Line #270:

```
# A simple customer list
customers = ["Robert", "Bryan", "John", "Jo", "Brittney"]

# Sort by last letter in name
customers.sort(key = lambda name: list(name)[-1])

# Display the results
print(customers)
```

Code Line #271:

```
['Bryan', 'John', 'Jo', 'Robert', 'Brittney']
```

Code Line #272:

```
a = lambda x, y, z: x + y + z
print(a(1, 2, 3))
```

Code Line #273:

```
lock = threading.Lock()

with lock:
     self.critical_variable += 1
```

Code Line #274:

```
Original SHA256 hash: eba3f65217714021800aeae3c651cd4132800711078648dc117abc0a1b956fbc
New SHA256 hash: 407fa327c98cfdf8b42003bd4f7702b58f546239d33dd9c836df374da7054691
```

Code Line #275:

- Windows: `certutil -hashfile FILE SHA256`
- macOS: `openssl dgst -sha256 FILE`
- Linux: `sha256sum FILE`

Code Line #276:

```
Month,Coffee,Tea,Hot Chocolate,Espresso
Jan,523,301,507,332
Feb,621,339,501,339
Mar,512,218,497,401
Apr,511,401,324,385
```

Code Line #277:

```
[['Month', 'Coffee', 'Tea', 'Hot Chocolate', 'Espresso'],
     ['Jan', '523', '301', '507', '332'],
     ['Feb', '621', '339', '501', '339'],
     ['Mar', '512', '218', '497', '401'],
     ['Apr', '511', '401', '324', '385']]
```

Code Line #278:

```
[
     ['Month', 'Coffee', 'Tea', 'Hot Chocolate', 'Espresso'],
     ['Jan', '523', '301', '507', '332'],
     ['Feb', '621', '339', '501', '339'],
     ['Mar', '512', '218', '497', '401'],
     ['Apr', '511', '401', '324', '385']
]
```

Code Line #279:

```
sales_data[1][2]
```

Code Line #280:

```
pip install PACKAGE
```

Code Line #281:

```
import py_compile
py_compile.compile("file.py")
```

Code Line #282:

```
'__pycache__/file.cpython-39.pyc'
```

Code Line #283:

```
import py_compile
py_compile.compile("file.py")
```

Code Line #284:

```
'__pycache__/file.cpython-39.pyc'
```

Code Line #285:

```
python -m compileall
```

Code Line #286:

```
200003 function calls in 0.161 seconds

Ordered by: standard name

Ncalls   tottime  percall cumtime percall filename:lineno(function)
50000    0.015    0.000    0.155    0.000    <stdin>:1(myfunction)
1        0.006    0.006    0.161    0.161    <string>:1(<module>)
50000    0.002    0.000    0.002    0.000    {built-in method builtins.abs}
1        0.000    0.000    0.161    0.161    {built-in method builtins.exec}
50000    0.135    0.000    0.135    0.000    {built-in method builtins.print}
50000    0.003    0.000    0.003    0.000    {built-in method math.cos}
1        0.000    0.000    0.000    0.000    {method 'disable' of '_lsprof.Profiler' objects}
```

Code Line #287:

```
# Avoid this
for i in range(100):
     print(math.sin(math.pi / 3))

# Instead, do this
a = math.sin(math.pi / 3)
for i in range(100):
     print(a)
```

Code Line #288:

```
**** SLOW FUNCTION ****
       1000004 function calls in 0.317 seconds

     Ordered by: standard name
     ncalls tottime percall cumtime  percall filename:lineno(function)
          1   0.000    0.000    0.317    0.317 <string>:1(<module>)
          1   0.208    0.208    0.317    0.317 cache-results.py:6(slow_func)
     100000   0.013    0.000    0.013    0.000 contextlib.py:329(__init__)
     100000   0.039    0.000    0.051    0.000 contextlib.py:334(__enter__)
     100000   0.024    0.000    0.035    0.000 contextlib.py:339(__exit__)
          1   0.000    0.000    0.317    0.317 {built-in method builtins.exec}
     100000   0.004    0.000    0.004    0.000 {built-in method builtins.getattr}
     100000   0.004    0.000    0.004    0.000 {built-in method builtins.print}
     200000   0.010    0.000    0.010    0.000 {built-in method builtins.setattr}
     100000   0.005    0.000    0.005    0.000 {built-in method math.sin}
     100000   0.004    0.000    0.004    0.000 {method 'append' of 'list' objects}
          1   0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     100000    0.006    0.000    0.006    0.000 {method 'pop' of 'list' objects}

**** FAST FUNCTION ****
       900005 function calls in 0.285 seconds

     Ordered by: standard name

     ncalls tottime percall cumtime  percall filename:lineno(function)
          1    0.000    0.000    0.285    0.285 <string>:1(<module>)
          1    0.188    0.188    0.285    0.285 cache-results.py:12(fast_func)
     100000    0.013    0.000    0.013    0.000 contextlib.py:329(__init__)
     100000    0.037    0.000    0.048    0.000 contextlib.py:334(__enter__)
     100000    0.023    0.000    0.033    0.000 contextlib.py:339(__exit__)
          1    0.000    0.000    0.285    0.285 {built-in method builtins.exec}
     100000    0.004    0.000    0.004    0.000 {built-in method builtins.getattr}
     100000    0.003    0.000    0.003    0.000 {built-in method builtins.print}
     200000    0.009    0.000    0.009    0.000 {built-in method builtins.setattr}
          1    0.000    0.000    0.000    0.000 {built-in method math.sin}
     100000    0.003    0.000    0.003    0.000 {method 'append' of 'list' objects}
          1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     100000    0.005    0.000    0.005    0.000 {method 'pop' of 'list' objects}
```

Code Line #289:

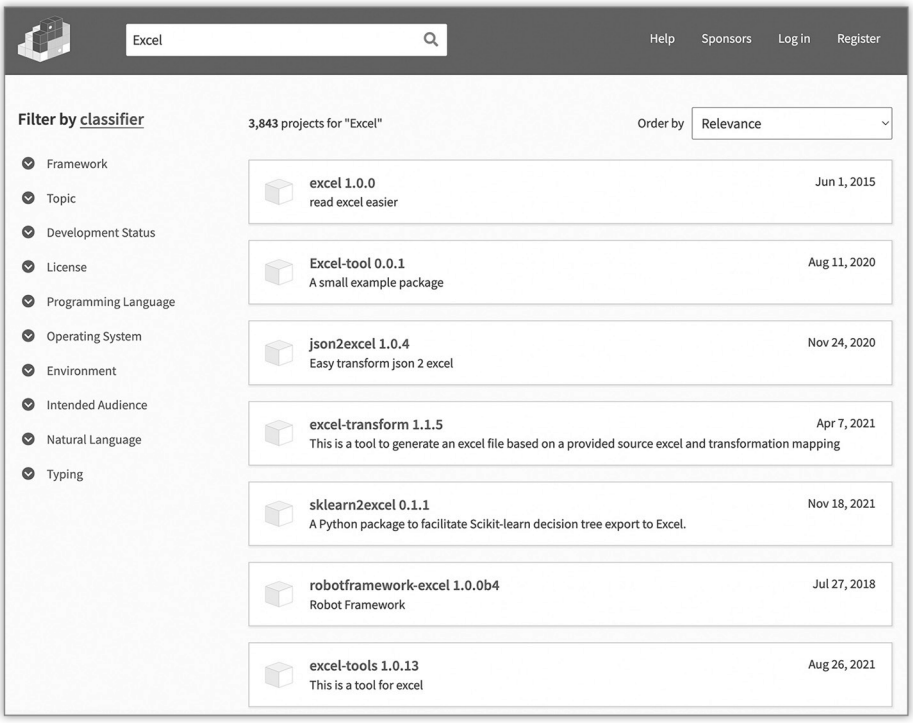
```
# Avoid this
a = 1
b = 2
c = 3
d = 4

# Instead, do this
a, b, c, d = 1, 2, 3, 4
```

Code Line #290:

```
**** SLOW FUNCTION ****
       4 function calls in 0.010 seconds
     Ordered by: standard name
     ncalls  tottime  percall  cumtime  percall filename:lineno(function)
          1    0.000    0.000    0.010    0.010 <string>:1(<module>)
          1    0.010    0.010    0.010    0.010 assignment-results.py:3(slow_func)
          1    0.000    0.000    0.010    0.010 {built-in method builtins.exec}
          1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
**** FAST FUNCTION ****
         4 function calls in 0.007 seconds
   Ordered by: standard name
     ncalls  tottime  percall  cumtime  percall filename:lineno(function)
          1    0.000    0.000    0.007    0.007 <string>:1(<module>)
          1    0.007    0.007    0.007    0.007 assignment-results.py:11(fast_func)
          1    0.000    0.000    0.007    0.007 {built-in method builtins.exec}
          1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

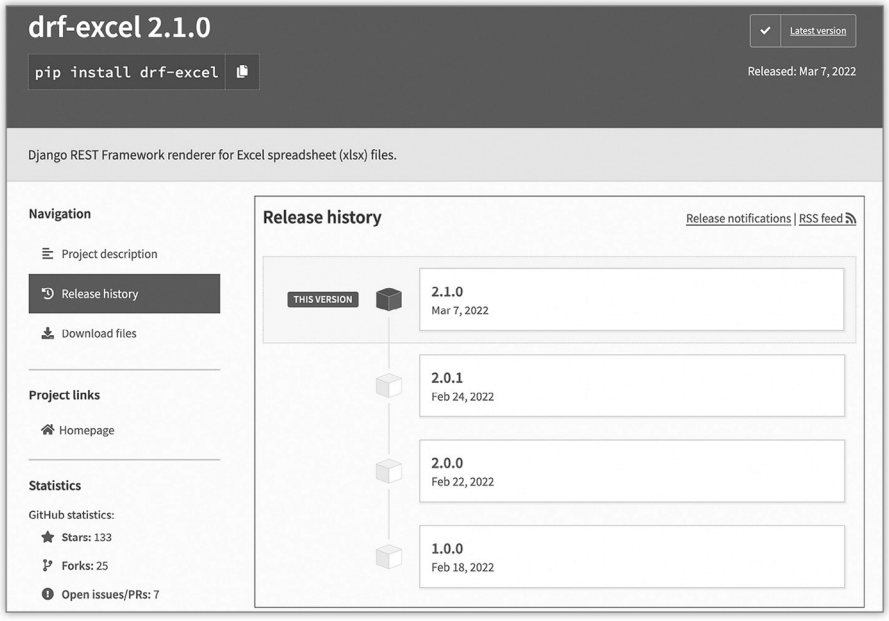
CHAPTER 21
What's Next?

fig. 53



The Python Package Index list of packages relating to "Excel."

fig. 54



A look at the release history of the drf-excel package.

Code Line #291:

```
[python] converting string to integer
```