Chapter 4

How to define and use functions and modules

Objectives

Applied

1. Define and use functions in your programs including the use of default values, named arguments, local variables, and global variables.
2. Create, document, import, and use your own modules.
3. Import and use the random module.
4. Use a hierarchy chart or outline to plan the functions of a program.

Knowledge

1. In general terms, describe how to define a function, including the use of a return statement.
2. In general terms, describe how to call a function.
3. In general terms, describe how to define and call a main() function.

Objectives

4. Describe the use of default values in a function definition and in the statements that call the function.
5. Describe the use of named arguments in calling statements.
6. Describe the recommended use of global variables, local variables, and global constants.
7. In general terms, explain how to create and document a module.
8. Distinguish among importing a module into the default namespace, a specified namespace, and the global namespace.
9. Describe how to use Python standard modules, such as the random module.
10. Describe the problem that can occur if you import a module into the global namespace.
11. Explain how to use a hierarchy chart or outline to plan the functions of a program.

MURACH BOOKS
© 2016, Mike Murach & Associates, Inc.

The syntax for defining a function

```
def function_name([arguments]):
    statements
```

A function that doesn’t accept arguments

How to define it

```
def print_welcome():
    print("Welcome to the Future Value Calculator")
    print()
```

How to call it

```
print_welcome()                 # prints welcome message
```

A function that has one argument

How to define it

```
def print_welcome(message):
    print(message)
    print()
```

How to call it

```
message = "Welcome to the Future Value Calculator"
print_welcome(message)           # prints welcome message
```

A function that accepts two arguments and returns a value

How to define it

```
def calculate_miles_per_gallon(miles_driven, gallons):
    mpg = miles_driven / gallons
    mpg = round(mpg, 1)
    return mpg
```

How to call it

```
miles = 500
gallons = 14
mpg = calculate_miles_per_gallon(miles, gallons)  # 35.7
```

A main() function that calls another function

```
#!/usr/bin/env python3

def main():
    # display a welcome message
    print("The Future Value Calculator\n")

    # get input
    monthly_investment = float(input(
        "Enter monthly investment:   "))
    yearly_interest = float(input(
        "Enter yearly interest rate: "))
    years = int(input(
        "Enter number of years:      "))

    # get the future value
    future_value = calculate_future_value(
        monthly_investment, yearly_interest, years)

    # display output
    print("Future value:              ", round(future_value, 2))
```

Two ways to call a main() function

Code a simple call statement (not recommended)

```
main()
```

Code a call statement within an if statement that checks if current module is main module

```
if __name__ == "__main__":   # if main module
    main()                   # call main() function
```

The user interface

```
Enter monthly investment:    100
Enter yearly interest rate: 12
Enter number of years:       10
Future value:                23233.91

Continue? (y/n):
```

The code

```
#!/usr/bin/env python3

def calculate_future_value(monthly_investment, yearly_interest,
                           years):
    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest / 12 / 100
    months = years * 12

    # calculate future value
    future_value = 0.0
    for i in range(0, months):
        future_value += monthly_investment
        monthly_interest = future_value * monthly_interest_rate
        future_value += monthly_interest
    return future_value
```

The code (cont.)

```
def main():
    choice = "y"
    while choice.lower() == "y":
        # get input from the user
        monthly_investment = float(input(
            "Enter monthly investment:\t"))
        yearly_interest_rate = float(input(
            "Enter yearly interest rate:\t"))
        years = int(input("Enter number of years:\t\t"))

        # get and display future value
        future_value = calculate_future_value(
            monthly_investment, yearly_interest_rate, years)

        print("Future value:\t\t\t"
            + str(round(future_value, 2)))
        print()
```

The code (cont.)

```
            # see if the user wants to continue
            choice = input("Continue? (y/n): ")
            print()

        print("Bye!")

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

A function with a default value

```
def calculate_future_value(monthly_investment,
                           yearly_interest, years=20):
    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest / 12 / 100
    months = years * 12

    # calculate future value
    future_value = 0.0
    for i in range(0, months):
        future_value += monthly_investment
        monthly_interest = future_value *
                           monthly_interest_rate
        future_value += monthly_interest

    return future_value
```

How to call the function and use its default value

```
future_value = calculate_future_value(100, 8.5)
```

How to call the function and override its default value

```
future_value = calculate_future_value(100, 8.5, 10)
```

Murach Books
© 2016, Mike Murach & Associates, Inc.

How to use default values in your function definitions

- You can specify a default value for any argument in a function definition by assigning a value to the argument. However, the arguments with default values must be coded last in the function definition.
- When you call a function, any arguments that have default values are optional. But you can override the default value for an argument by supplying that argument.

How to call the function with named arguments

```
future_value = calculate_future_value(
    years=10,
    monthly_investment=100,
    yearly_interest=8.5)
```

How to use named arguments in your calling statements

- To code a *named argument*, code the name of the argument in the function definition, an equals sign, and the value or variable for the argument.
- If you call a function without named arguments, you must code them in the same sequence that they're coded in the function definition.
- If you call a function with named arguments, you don't have to code the arguments in the sequence that they're coded in the function definition.
- It's a good practice to use named arguments for functions that have many arguments. This can improve the readability of the code and reduce errors.

Functions that use local variables

```
def calc_tax(amount, tax_rate):
    tax = amount * tax_rate          # tax is local variable
    return tax                       # return is necessary

def main():
    tax = calc_tax(85.0, .05)        # tax is local variable
    print("Tax:", tax)               # Tax 4.25
```

A function that changes a global variable (not recommended)

```
tax = 0.0                           # tax is global variable

def calc_tax(amount, tax_rate):
    global tax                      # access global variable
    tax = amount * tax_rate         # change global variable

def main():
    calc_tax(85.0, .05)
    print("Tax:", tax)              # Tax 4.25 (global)
```

A local variable that shadows a global variable (not recommended)

```
tax = 0.0                             # tax is global variable

def calc_tax(amount, tax_rate):
    tax = amount * tax_rate           # tax is local variable
    print("Tax:", tax)                # Tax 4.25 (local)

def main():
    calc_tax(85.0, .05)
    print("Tax:", tax)                # Tax 0.0 (global)
```

A function that uses a global constant (okay)

```
TAX_RATE = 0.05                     # TAX_RATE is global
def calc_tax(amount):
    tax = amount * TAX_RATE         # use constant here
    return tax
```

MODULES

The temperature.py file (temperature module)

```
def to_celsius(fahrenheit):
    celsius = (fahrenheit - 32) * 5/9
    return celsius

def to_fahrenheit(celsius):
    fahrenheit = celsius * 9/5 + 32
    return fahrenheit

# the main() function is used to test the other functions
# this code isn't run if this module isn't the main module
def main():
    for temp in range(0, 212, 40):
        print(temp, "Fahrenheit =",
              round(to_celsius(temp)), "Celsius")

    for temp in range(0, 100, 20):
        print(temp, "Celsius =",
              round(to_fahrenheit(temp)), "Fahrenheit")

# if this module is the main module, call the main() function
# to test the other functions
if __name__ == "__main__":
    main()
```

The console when you run this module

```
0 Fahrenheit = -18 Celsius
40 Fahrenheit = 4 Celsius
80 Fahrenheit = 27 Celsius
120 Fahrenheit = 49 Celsius
160 Fahrenheit = 71 Celsius
200 Fahrenheit = 93 Celsius
0 Celsius = 32 Fahrenheit
20 Celsius = 68 Fahrenheit
...
```

The temperature module with documentation

```
"""
This module contains functions for converting
temperature between degrees Fahrenheit
and degrees Celsius
"""
def to_celsius(fahrenheit):
    """
    Accepts degrees Fahrenheit (fahrenheit argument)
    Returns degrees Celsius
    """
    celsius = (fahrenheit - 32) * 5/9
    return celsius

def to_fahrenheit(celsius):
    """
    Accepts degrees Celsius (celsius argument)
    Returns degrees Fahrenheit
    """
    fahrenheit = celsius * 9/5 + 32
    return fahrenheit
```

How to view the documentation for a module

```
>>> import temperature
>>> help(temperature)
Help on module temperature:

NAME
    temperature

DESCRIPTION
    This module contains functions for converting
    temperature between degrees Fahrenheit
    and degrees Celsius

FUNCTIONS
    to_celsius(fahrenheit)
        Accepts degrees Fahrenheit (fahrenheit argument)
        Returns degrees Celsius

    to_fahrenheit(celsius)
        Accepts degrees Celsius (celsius argument)
        Returns degrees Fahrenheit
```

The syntax for importing a module into a local namespace

```
import module_name [as namespace]
```

Importing into the module's default namespace

```
import temperature
```

Code that calls its functions

```
c = temperature.to_celsius(f)
f = temperature.to_fahrenheit(c)
```

Importing into a specified namespace

```
import temperature as temp
```

Code that calls its functions

```
c = temp.to_celsius(f)
f = temp.to_fahrenheit(c)
```

The syntax for importing into the global namespace (not recommended)

```
from module_name import function_name1[,
function_name2]...
```

Importing one function into the global namespace

```
from temperature import to_celsius
```

Code that calls its functions

```
c = to_celsius(f)
f = to_fahrenheit(c)    # Error! Function not imported
```

Importing all functions into the global namespace

```
from temperature import *
```

Code that calls its functions

```
c = to_celsius(f)
f = to_fahrenheit(c)
```

The user interface

```
MENU
1. Fahrenheit to Celsius
2. Celsius to Fahrenhit

Enter a menu option: 1
Enter degrees Fahrenheit: 99
Degrees Celsius: 37.2

Convert another temperature? (y/n): n
```

The code

```
import temperature as temp

def display_menu():
    print("MENU")
    print("1. Fahrenheit to Celsius")
    print("2. Celsius to Fahrenhit")
    print()

def convert_temp():
    option = int(input("Enter a menu option: "))
    if option == 1:
        f = int(input("Enter degrees Fahrenheit: "))
        c = temp.to_celsius(f)
        c = round(c, 2)
        print("Degrees Celsius:", c)
    elif option == 2:
        c = int(input("Enter degrees Celsius: "))
        f = temp.to_fahrenheit(c)
        f = round(f, 2)
        print("Degrees Fahrenheit:", f)
    else:
        print("You must enter a valid menu number.")
```

The code

```
def main():
    display_menu()
    again = "y"
    while again.lower() == "y":
        convert_temp()
        print()
        again = input("Convert another temperature? (y/n): ")
        print()
    print("Bye!")

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

Some of the standard modules presented in this book

```
math
random
decimal
csv
pickle
tkinter
```

Three functions of the random module

```
random()
randint(min, max)
randrange([start,] stop [,step])
```

A statement that imports the random module

```
import random
```

Examples that use the random module

```
# the use of the random method
number = random.random()       # float >= 0.0 and < 1.0
number = random.random() * 100 # float >= 0.0 and < 100.0

# the use of the randint method
number = random.randint(1, 100)   # int from 1 to 100
number = random.randint(101, 200) # int from 101 to 200
number = random.randint(0, 7)     # int from 0 to 7

# the use of the randrange method
number = random.randrange(1, 100)       # int from 1 to 99
number = random.randrange(
             100, 200, 2)      # even int from 100 to 198
number = random.randrange(
             11, 250, 2)       # odd int from 11 to 249
```

Code that simulates rolling a pair of dice

```
die1 = random.randint(1, 6)    # assume 6 is returned
die2 = random.randint(1, 6)    # assume 5 is returned
print("Your roll:", die1, die2)    # Your roll: 6 5
```

The user interface

```
Guess the Number!

I'm thinking of a number from 1 to 10

Your guess: 5
Too low.
Your guess: 8
You guessed it in 2 tries.

Would you like to play again? (y/n): n

Bye!
```

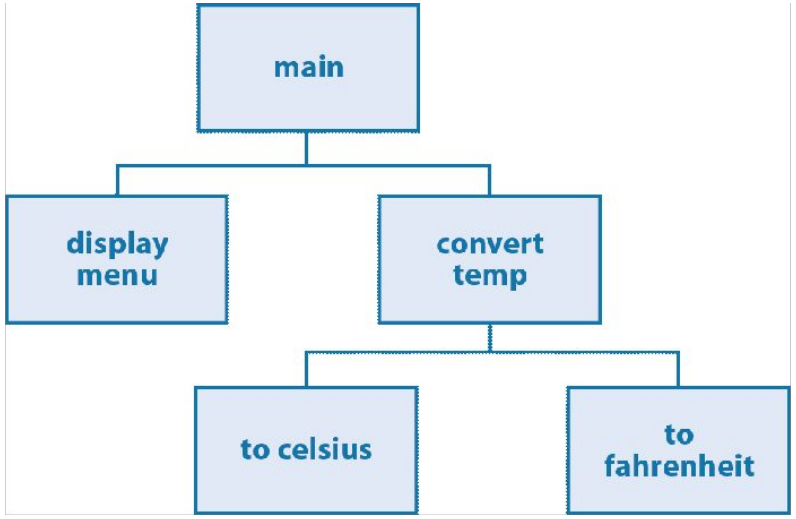
The code

```
import random

LIMIT = 10

def display_title():
    print("Guess the number!\n")

def play_game():
    number = random.randint(1, LIMIT)
    print("I'm thinking of a number from 1 to "
        + str(LIMIT) + "\n")
    count = 1
    while True:
        guess = int(input("Your guess: "))
        if guess < number:
            print("Too low.")
            count += 1
        elif guess > number:
            print("Too high.")
            count += 1
        elif guess == number:
            print("You guessed it in "
                + str(count) + " tries.\n")
            return
```

The code (cont.)

```
def main():
    display_title()
    again = "y"
    while again.lower() == "y":
        play_game()
        again = input("Would you like to play again? (y/n): ")
        print()
    print("Bye!")

# if started as the main module, call the main() function
if __name__ == "__main__":
    main()
```

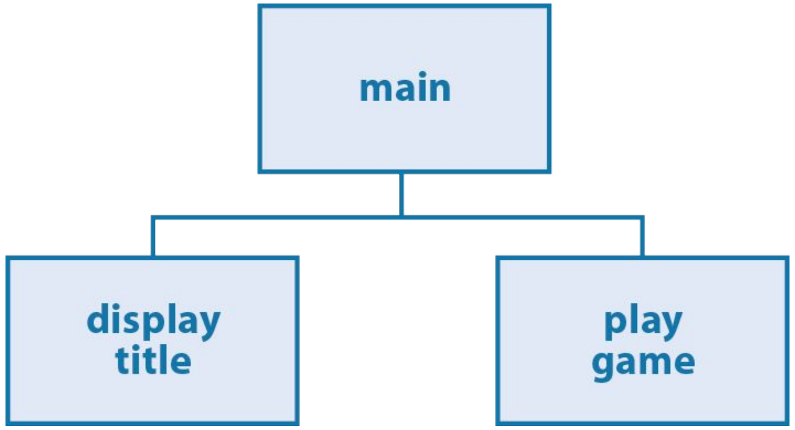
A hierarchy chart for the Convert Temperatures program



A hierarchy outline for the same program

```
main
    display menu
    convert temp
        to celsius
        to fahrenheit
```

A hierarchy chart for the Guess the Number game



How to build a hierarchy chart

- Start with a box for the main() function.
- At the next level, put boxes for the functions that the main() function needs to call. This usually includes the function that will control the main action of the program, plus any functions that need to be done before or after that function.
- Continue down the levels by dividing the higher-level functions into their component functions until there aren't any more components.

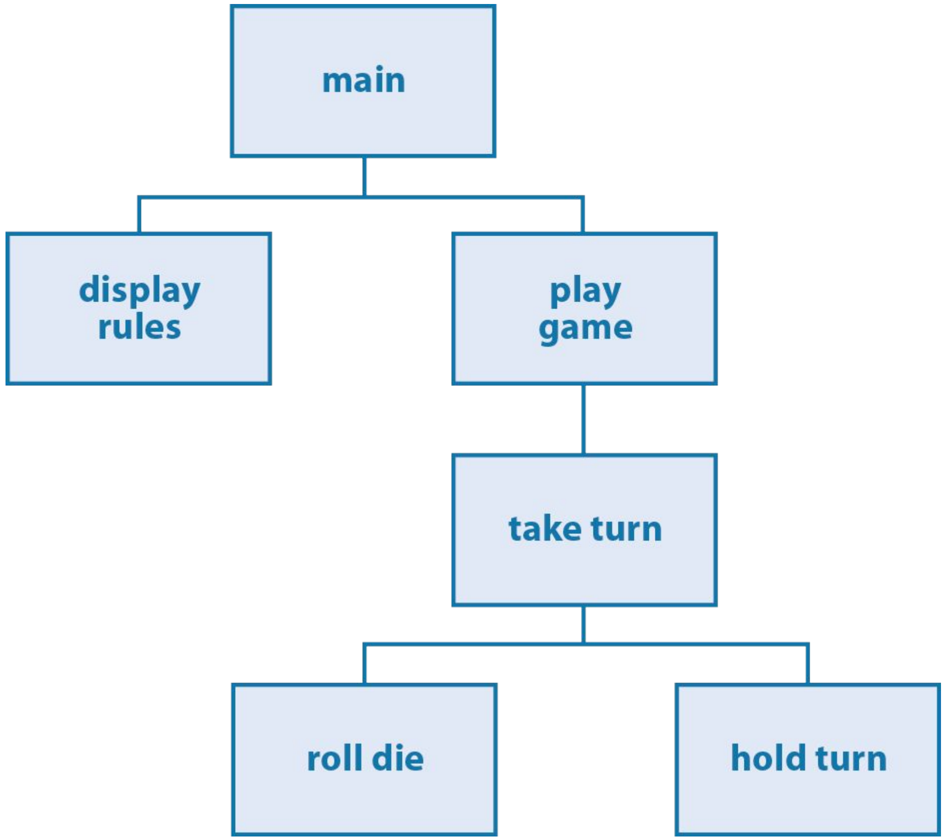
Guidelines for creating hierarchy charts

- The names in a chart should start with a verb and give a good indication of what the function does. Then, the names can easily be converted to Python function names.
- Each function should do everything that is related to the function name and nothing more. The code shouldn't be split between two or more functions.

The rules for a Pig Dice game

- For each turn, the player can choose to roll the die or hold.
- A turn ends when the player rolls a 1 or chooses to hold.
- If the player rolls a 1, all points are lost for that turn.
- If the player chooses to hold, all points for that turn are added to the total.
- If the player enters an invalid character, the roll or hold prompt is displayed and the player can make another entry.
- When a player reaches 20 points, the game ends and the number of turns is displayed.

A hierarchy chart for the Pig Dice game



The user interface

```
TURN 1
Roll or hold? (r/h): r
Die: 4
Roll or hold? (r/h): h
Score for turn: 4
Total score: 4

TURN 2
Roll or hold? (r/h): r
Die: 1
Turn over. No score.
...
...
You finished in 4 turns!
```

The code with global variables

```
import random

# global variables
turn = 1
score = 0
score_this_turn = 0
turn_over = False
game_over = False

def main():
    display_rules()
    play_game()

def display_rules():
    print("Let's Play PIG!")
    print()
    print("* See how many turns it takes you to get to 20.")
    print("* Turn ends when you hold or roll a 1.")
    print("* If you roll a 1, you lose all points for the turn.")
    print("* If you hold, you save all points for the turn.")
    print()
```

The code with global variables (cont.)

```
def play_game():
    while not game_over:
        take_turn()
    print()
    print("Game over!")

def take_turn():
    global turn_over

    print("TURN", turn)
    turn_over = False
    while not turn_over:
        choice = input("Roll or hold? (r/h): ")
        if choice == "r":
            roll_die()
        elif choice == "h":
            hold_turn()
        else:
            print("Invalid choice. Try again.")
```

The code with global variables (cont.)

```
def roll_die():
    global turn, score_this_turn, turn_over
    die = random.randint(1, 6)
    print("Die:", die)
    if die == 1:
        score_this_turn = 0
        turn += 1
        print("Turn over. No score.\n")
        turn_over = True
    else:
        score_this_turn += die

def hold_turn():
    global turn, score_this_turn, score, turn_over, game_over
    print("Score for turn:", score_this_turn)
    score += score_this_turn
    score_this_turn = 0
    print("Total score:", score, "\n")

    turn_over = True
    if score >= 20:
        game_over = True
        print("You finished in", turn, "turns!")
    turn += 1
```

The code with global variables (cont.)

```
# if started as the main module, call the main() function
if __name__ == "__main__":
    main()
```

The functions with local variables instead

```
def play_game():
    turn = 1
    score = 0
    game_over = False
    while not game_over:
        turn, score, game_over = take_turn(
            turn, score, game_over)
    print()
    print("Game over!")
```

The functions with local variables instead (cont.)

```
def take_turn(turn, score, game_over):
    print("TURN", turn)
    score_this_turn = 0
    turn_over = False
    while not turn_over:
        choice = input("Roll or hold? (r/h): ")
        if choice == "r":
            turn, score, score_this_turn, turn_over = \
                roll_die(turn, score, score_this_turn)
        elif choice == "h":
            turn, score, turn_over, game_over =       \
                hold_turn(turn, score, score_this_turn)
        else:
            print("Invalid choice. Try again.")
    return turn, score, game_over
```

The functions with local variables instead (cont.)

```
def roll_die(turn, score, score_this_turn):
    die = random.randint(1, 6)
    print("Die:", str(die))
    if die == 1:
        score_this_turn = 0
        turn += 1
        print("Turn over. No score.\n")
        turn_over = True
    else:
        score_this_turn += die
        turn_over = False
    return turn, score, score_this_turn, turn_over
```

The functions with local variables instead (cont.)

```
def hold_turn(turn, score, score_this_turn):
    print("Score for turn:", score_this_turn)
    score += score_this_turn
    print("Total score:", score, "\n")
    turn_over = True
    game_over = False
    if score >= 20:
        print("You finished in", turn, "turns!")
        game_over = True
        return turn, score, turn_over, game_over
    turn += 1
    return turn, score, turn_over, game_over
```