Functions Part 2

Section 2

Chapter 4

Quiz 9

Functions Calling Other Functions

- Function can call another function

- When the called function terminates
  - Control returns to the place in calling function just after where function call occurred.

Example 1: functions calling other functions

```
def main():
    firstPart()
    secondPart()

def firstPart():
    print("First Part")

def secondPart():
    print("Second Part")

main()
```

```
First Part
Second Part
```

Functions Returning Multiple Values

- Functions can return any type of object, not just a number, string, or Boolean value.

- For instance, a function can return a tuple or a list of numbers.

Example 2

```
(4, 5, 6)
```

```
def main():
    print(plus_minus(5))

def plus_minus(x):
    return x-1, x, x+1

main()
```

List Comprehension

- Simpler way to apply a certain function to each item of a list
  - Use list comprehension

```
list2 = [f(x) for x in list1]
```

- The *for* clause in a list comprehension can optionally be followed by an *if* clause.

```
[g(x) for x in list1 if int(x) % 2 == 1]
```

Example 3: List Comprehension

```
# list comprehension

lst1 = [1.2, 2.5, 3.5, 4.6]
print(lst1)

lst2 = [int(x) for x in lst1]
print(lst2)
```

```
[1.2, 2.5, 3.5, 4.6]
[1, 2, 3, 4]
```

Example 4: List Comprehension

```
lst = [i*2 for i in range(5)]
print(lst)
```

```
[0, 2, 4, 6, 8]
```

Recall: Classwork 9

```
#Classwork 9: Present value of ordinary annuity
#Name:
#SID:

#list comprehension
def presentValue(c, r, n):
    return round(sum([c/(1+r)**i for i in range(1, n+1)]),2)
```

Default Values

- Parameters of a function can have default values
  - Assigned to them when no values are passed to them

- Format for definition using default values

```
def functionName(par1, par2, par3=value3, par4=value4):
```

Default Values

```
>>> #recall the range() function
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(1,5,2))
[1, 3]
```

```
def total(w, x, y=10, z=20):
    return (w ** x) + y + z
```

TABLE 4.4 **Three function calls.**

Function Call	Value	Calculated As
`total(2, 3)`	38	$2^3 + 10 + 20$
`total(2, 3, 4)`	32	$2^3 + 4 + 20$
`total(2, 3, 4, 5)`	17	$2^3 + 4 + 5$

Passing by Parameter Name

- Arguments can be passed to functions by using names of the corresponding parameters
  - Instead of relying on position

```
def difference(x, y):
    return x-y

print(difference(x=2, y=3))
print(difference(y=3, x=2))
```

```
==:
-1
-1
```

Lambda Expressions

- One-line mini-functions
  - Can be used where a simple function is required.
  - Compute a single expression
  - Cannot be used as a replacement for complex functions

- Format
  - Where *expression* is the value to be returned

```
lambda par1, par2, ...: expression
```

Lambda Expressions

```
9
```

```
#lambda expressions (One-line mini-functions)

s = lambda s: s**2

print(s(3))
```

Lambda Expressions

```
#lambda expressions (One-line mini-functions)

s = lambda s: s**2

print(s(3))
```

Same output (9)

```
>>>
========
9
9
>>>
```

```
#regular function approach
def getSquare(s):
    return s**2
print(getSquare((3)))
```

Recursion

Section 4

Chapter 6

A Recursive Power Function

- Recursive function invokes/calls itself
  - Successive calls reduce to simpler task
  - Until base case with trivial solution reached
- The n^{th} power of a number
  - Iteratively

$$r^n = \underbrace{r \cdot r \cdot \cdots \cdot r}_{n \text{ terms}}$$

  - Recursively

$$r^1 = r$$
$$r^n = r \cdot r^{n-1}$$

A Recursive Power Function

- Example 1: Definition uses the iterative definition of a power.

Example 1 **Power Function** The following program uses the iterative definition of a *power* function. The function definition requires two temporary variables (*value* and *i*). Also, the function definition does not resemble the iterative definition above.

```
def power(r, n):
    ## iterative definition of power function
    value = 1
    for i in range(1, n + 1):
        value = r * value
    return value

print(power(2, 3))
```

[Run]

```
8
```

A Recursive Power Function

- Example 2: Definition uses the recursive definition of a power.

```
def power(r,n):
    if n == 1:
        return r
    else:
        return r* power(r, n-1)

print(power(2,3))
```

A Recursive Power Function

Traits of recursive algorithms

1. One or more base cases with direct solutions.
2. An "inductive step"
   - Reducing the problem to one or more smaller versions of the same problem
   - Reduction eventually culminating in a base case.
   - Called the reducing step.

A Recursive Power Function

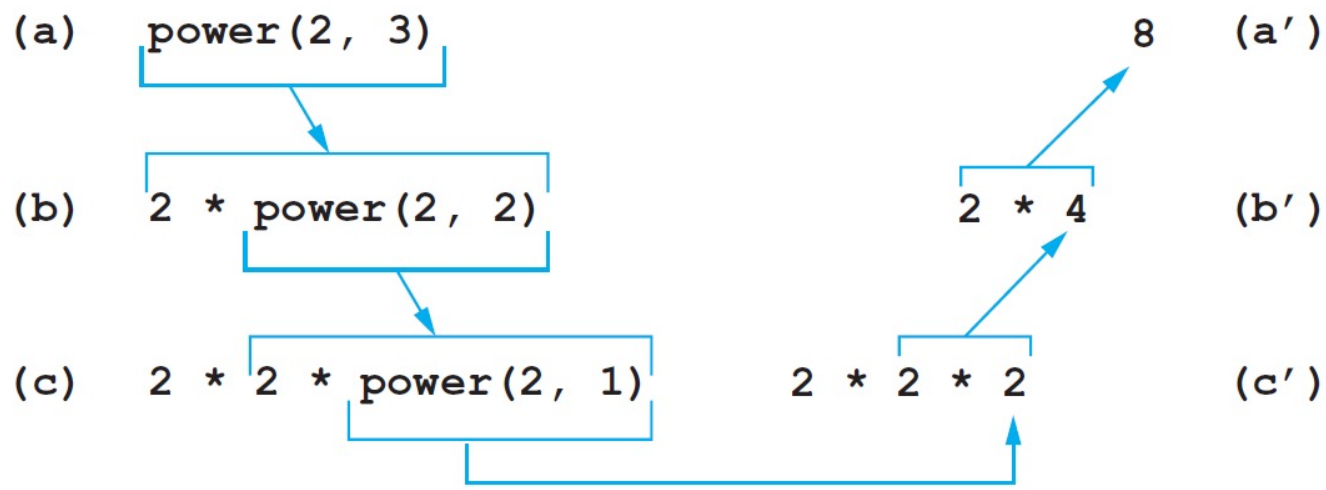

FIGURE 6.20 The recursive computation of power(2, 3).

Recursive Palindrome Function

- Sometimes a recursive solution is easier to understand and code than iterative routine.

- Function uses recursion to determine whether or not word is a palindrome. A word is a palindrome if it reads the same forward and backward, e.g., racecar, kayak, and pullup

Example 3 **Palindrome** The following function uses recursion to determine whether or not a word containing no punctuation is a palindrome.

```
def isPalindrome(word):
    word = word.lower()        # Convert all letters to lowercase.
    if len(word) <= 1:         # Words of zero or one letters are palindromes.
        return True
    elif word[0] == word[-1]: # First and last letters match.
        word = word[1:-1]      # Remove first and last letters.
        return isPalindrome(word)
    else:
        return False
```

Program Design

Section 3

Chapter 4

Top-Down Design

- To make a complicated problem more understandable
  - Divide it into smaller, less complex subproblems.
  - Called stepwise refinement

- Top-down design and structured programming
  - Techniques to enhance programming productivity

Top-Down Design

Criteria

1. Design should be easily readable and emphasize small module size.
2. Tasks proceed from general to specific as you read down the chart.
3. Subtasks should be single-minded.
4. Subtasks should be independent of each other.

Top-Down Design

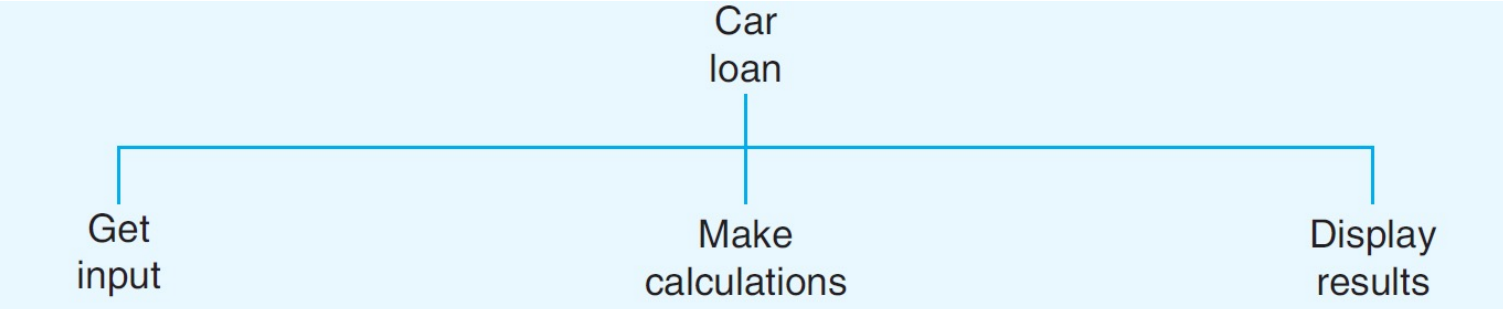

FIGURE 4.33 Beginning of a hierarchy chart for the car loan program.

Top-Down Design

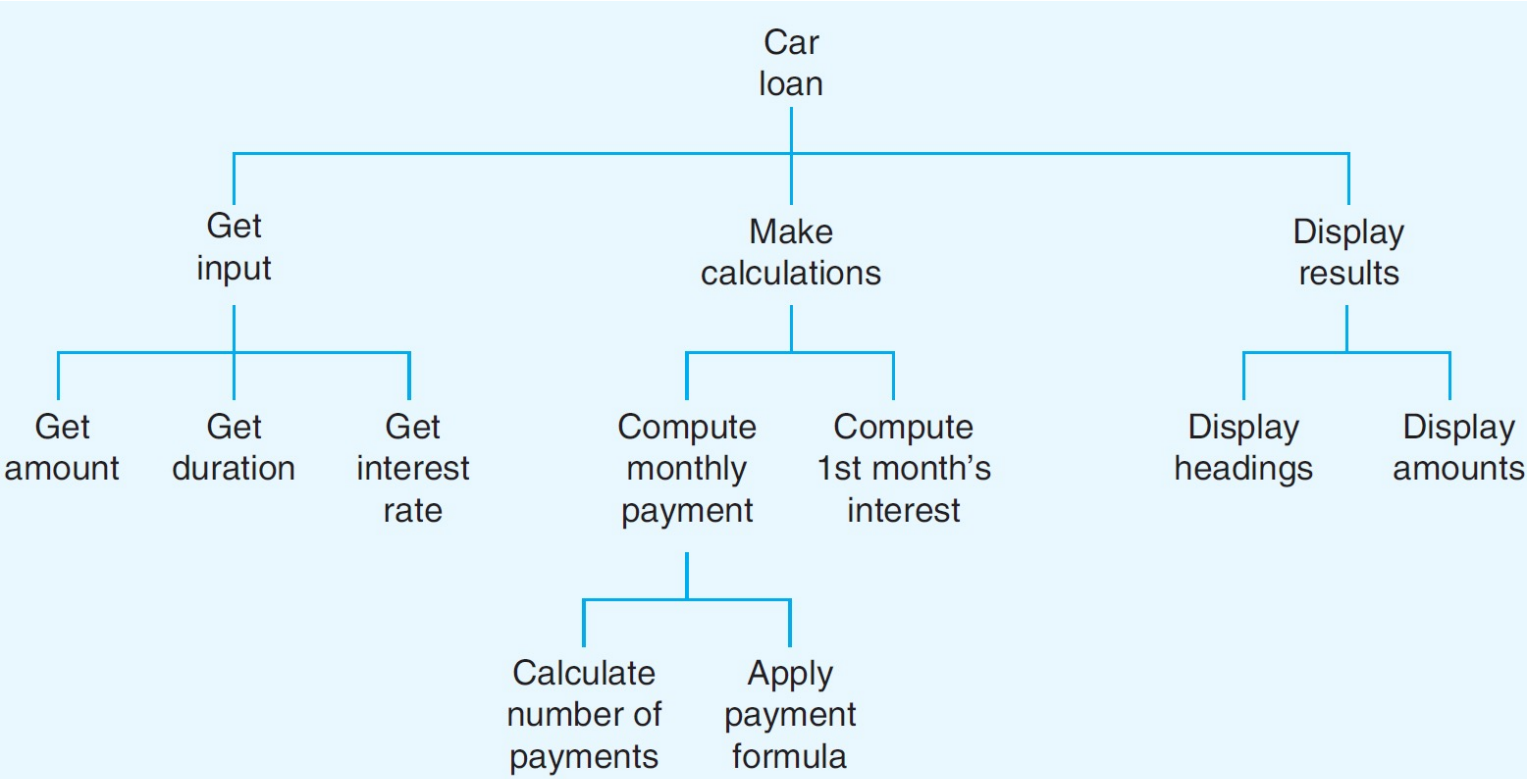

FIGURE 4.34 Hierarchy chart for the car loan program.

Structured Programming

- A program is structured if it meets modern standards of program design
- Use top-down design
- Use only the three types of logical structures:
  - Sequences – Statements executed one after the other
  - Decisions – blocks of code executed based on test of some condition
  - Loops – blocks of coded executed repeatedly based on some condition

Advantages of Structured Programming

- Easy to write
- Easy to debug
- Easy to understand
- Easy to change

Object-Oriented Programming

- An object is an encapsulation of data and code that operates on the data

- Objects
  - Have properties
  - Respond to methods
  - Raise events

- Object-oriented program viewed as collection of cooperating objects

Classwork 9. Recursion

- Suppose that the sum function for lists does not exist. Write a recursive function recur_sum(lst) that totals the numbers in a list of numbers. The function should be capable of the following. Upload the .py file and the output screenshot on Canvas.

```
>>> recur_sum([2, 3])
5
>>> recur_sum([2, 3, 7])
12
>>> recur_sum([2, 3, 7, 5])
17
>>> recur_sum([2, 3, 7, 5, 2])
19
```