EXERCISES

1. Determine the value of each of these expressions:

   (a) `1 + 2 * 3 - 4 * 5 + 6`
   (b) `1 + 2 ** 3 * 4 - 5`
   (c) `7 // 3`
   (d) `15 // 4`
   (e) `1.4**2 > 2`
   (f) `12 <= 3**3`
   (g) `7 % 3`
   (h) `244 % 2`
   (i) `floor(15.4)`
   (j) `ceil(8.229)`

2. Determine the value of each of these expressions:

   (a) `1 * 2 - 3 * 4 + 5 - 6`
   (b) `1 / 2 - 3 / 4`
   (c) `6 // 8`
   (d) `10 // 12`
   (e) `6//7 == 1//7`
   (f) `5/8 != 5//8`
   (g) `6 % 8`
   (h) `1011 % 2`
   (i) `ceil(23.18)`
   (j) `floor(-20.48)`

3. Determine the value of each of these expressions:

   (a) `(1 + 2) * (3 - 4) * 5 + 6`
   (b) `1 * 4 - 2 / 2`
   (c) `11 // 2`
   (d) `5 / 2 - 5 // 2`
   (e) `10 >= 11 or 11 < 12`
   (f) `7/2 == 3 or 1 >= 8/5`
   (g) `11 % 2`
   (h) `988 % 5`
   (i) `floor(-7.03)`
   (j) `ceil(-17.05)`

4. Determine the value of each of these expressions:

   (a) `1 - 2 * 3 + (4 + 5 * 6)`
   (b) `(1 + 4 - 2) / 2`
   (c) `9 // 7`
   (d) `23 // 6 + 14 // 3`
   (e) `3 > 1 and 4.25 > 9/2`
   (f) `1 == 5//3 and not 18 < 11`
   (g) `9 % 7`
   (h) `1543 % 10`
   (i) `ceil(-4.999)`
   (j) `floor(5.661)`

5. List the operations in the order they are performed in the expression `p*(1 + r)**t`.

6. List the operations in the order they are performed in the expression `p*1+r**t`.

7. Write a Python expression to compute $m = \dfrac{y2 - y1}{x2 - x1}$.

8. Write a Python expression to compute $y = ax^2 + bx + c$.

14. Change the perspective in the previous exercise by displaying, for each time period from 1 year to 10 years, the balance earned by each interest rate from 1% to 10%.

15. Write a program to print a table of values of n and 2^n for $n = 1, 2, \ldots, 10$. You do not need any functions other than `main()`.

16. Write a program to print a table of values of n, $\log n$, n^2, and 2^n for $n = 10, 20, \ldots, 200$. You do not need any functions other than `main()`.

17. Write a program to print a table of the areas of circles of radius $r = 1, 2, \ldots, 10$. Write a function to compute the area, which is πr^2.

18. Write a program to print a table of the volumes of spheres of radius $r = 1, 2, \ldots, 10$. Write a function to compute the volume, which is $\frac{4}{3}\pi r^3$.

19. Write a program to print a table of Fahrenheit to Celsius conversions for temperatures between $-30°F$ and $100°F$ at 10-degree intervals. Write a function to convert Fahrenheit to Celsius; the formula is $\frac{5}{9}(F - 32)$.

20. Write a program to print a table of mile-to-kilometer conversions for distances between 100 and 1500 miles at 100-mile intervals. Write a function to do the conversion; one mile is approximately 1.609 km.

21. The formula $208 - 0.7y$ is an estimate of a person's maximum heart rate in beats per minute when they are y years old. Write a program to print a table of values of these estimates for ages between 20 and 60 at 2-year intervals. Write a function to compute the estimate.

22. (Requires Exercise 21.) Write a function to return a description of a person's training zone based on his or her age and training heart rate, *rate*. The zone is determined by comparing *rate* with the person's maximum heart rate m: a rate at least 90% of m is considered `"interval training"`; otherwise, 70% of m is `"threshold training"`; and 50% of m is `"aerobic training"`. Any rate below 50% is classified as `"couch potato"`. Use the function from the previous exercise to estimate m, and write a `main()` to ask the user for input and display the result.

23. One way to approximate $\sqrt{x}$ is to start with an initial guess g and then repeatedly replace g with the average of g and x/g. Use this method to approximate $\sqrt{3}$ by hand.

24. Implement the idea from the previous exercise as a function `mysqrt(x)`. Use $g = x/2$ as the initial guess, and repeat until g^2 is within 10^{-10} (written `1e-10` in Python) of x. Compare your function with the `sqrt()` function.

EXERCISES

1. Determine the final value of the `result` accumulator after each of these accumulation loops:

   (a)
   ```
   result = 0
   for _ in range(5):
       result += 1
   ```
   (b)
   ```
   result = 0
   for i in range(5):
       result += i
   ```

2. Determine the final value of the `result` accumulator after each of these accumulation loops:

   (a)
   ```
   result = 0
   for _ in range(5):
       result += 2
   ```
   (b)
   ```
   result = 0
   for i in range(5):
       result += 2*i
   ```

3. Show the output of each of these code fragments:

   (a)
   ```
   result = 0
   for i in range(5):
       print(i, result)
       result += 2
   ```
   (b)
   ```
   result = 1
   for j in range(5):
       result *= 2
       print(j, result)
   ```

4. Show the output of each of these code fragments:

   (a)
   ```
   result = 0
   for j in range(1, 10, 2):
       result += j
       print(j, result)
   ```
   (b)
   ```
   result = 0
   for k in range(5):
       print(k, result)
       result += 2 * k + 1
   ```

5. Write an accumulation loop to compute each of these quantities:

   (a) $1 + 2 + 3 + \cdots + 100$

   (b) $5 + 10 + 15 + \cdots + 100$

   (c) $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{100}$

6. Write an accumulation loop to compute each of these quantities:

   (a) $2 + 4 + 6 + \cdots + 100$

   (b) $1 + 4 + 9 + 16 + \cdots + 100$

   (c) $1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{100}$

7. Write an accumulation loop to compute $20! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot 20$.

8. Write an accumulation loop to compute $2^{20} = 2 \cdot 2 \cdot 2 \cdot \cdots \cdot 2$.

11. Write a function `triangular(n)` to return the sum $1 + 2 + 3 + \cdots + n$. (These sums are known as triangular numbers.) Write a program to print a table of values of the triangular numbers for $n = 1, 2, 3, \ldots, 20$.

12. Write a function `harmonic(n)` to return the sum $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. (These sums are known as harmonic numbers.) Write a program to print a table of values of the harmonic numbers for $n = 1, 2, 3, \ldots, 20$.

13. Write a function `sum_of_evens(n)` to return the sum of the first n even integers, $2 + 4 + 6 + \cdots + 2n$. Write a program to print a table of values of this function for $n = 1, 2, 3, \ldots, 20$.

14. Write a function `sum_of_odds(n)` to return the sum of the first n odd integers, $1 + 3 + 5 + \cdots + (2n - 1)$. Write a program to print a table of values of this function for $n = 1, 2, 3, \ldots, 20$.

15. Write a function `factorial(n)` to return $n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$. Write a program to print a table of values of `factorial()` for $n = 1, 2, 3, \ldots, 20$.

16. Write a function `pyramid(n)` to return the sum $1^2 + 2^2 + 3^2 + \cdots + n^2$. (These sums are known as square pyramidal numbers.) Write a program to print a table of values of the pyramidal numbers for $n = 1, 2, 3, \ldots, 20$.

17. Write a function `baselsum(n)` to return the sum $1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2}$. Write a program to print a table of values of these sums for $n = 1, 2, 3, \ldots, 20$.

EXERCISES

1. Determine the final value of the `result` accumulator after each of these accumulation loops:

   (a)
   ```
   result = 0
   while result < 10:
       result += 3
   ```
   (b)
   ```
   result = 2
   while result < 10:
       result += result
   ```

2. Determine the final value of the `result` accumulator after each of these accumulation loops:

   (a)
   ```
   result = 10
   while result > 0:
       result -= 3
   ```
   (b)
   ```
   result = 0
   while result < 10:
       result += result + 1
   ```

3. Show the output of each of these code fragments:

   (a)
   ```
   i = 0
   result = 0
   while result < 10:
       i += 1
       result += i
       print(i, result)
   ```
   (b)
   ```
   j = 1
   result = 0
   while result < 10:
       result += j
       j *= 2
       print(j, result)
   ```

4. Show the output of each of these code fragments:

   (a)
   ```
   n = 1
   result = 0
   while result < 10:
       result += n
       n += 2
       print(n, result)
   ```
   (b)
   ```
   k = 1
   result = 0
   while k < 10:
       result += k
       k *= 3
       print(k, result)
   ```

5. Write an accumulation loop using `while` to compute each of these quantities:

   (a) $1 + 2 + 3 + \cdots + 100$

   (b) $5 + 10 + 15 + \cdots + 100$

   (c) $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{100}$

6. Write an accumulation loop using `while` to compute each of these quantities:

   (a) $2 + 4 + 6 + \cdots + 100$

   (b) $1 + 4 + 9 + 16 + \cdots + 100$

   (c) $1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{100}$

7. Write an accumulation loop using `while` to compute $20! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot 20$.

8. Write an accumulation loop using `while` to compute $2^{20} = 2 \cdot 2 \cdot 2 \cdot \cdots \cdot 2$.

9. Give another test that could be used in the `while` loop example on page 60 to produce the sequence 2, 6, 10, . . . , 50.

12. Modify the `years_to_goal()` function to add a parameter for an annual deposit. Add the deposit each year after computing interest. Write a complete program to ask the user for the principal, interest rate, annual deposit, and goal, and then print the number of years until the goal is reached.

13. A loan is taken out with interest compounded annually, where equal annual payments will be made.

    (a) Write a `years_to_payoff(`*`owed, rate, payment`*`)` function to return the number of years that it will take to pay off the loan. Use it to write a program that asks the user for the amount owed, interest rate, and payment amount, and then prints the number of years it will take to pay.

    (b) If the annual payment is not greater than the amount of interest for the first year, the loan will never be paid. Add a test to report this to the user instead of calling the function.

14. A loan is taken out with interest compounded monthly, where equal monthly payments will be made.

    (a) Write a `years_to_payoff(`*`owed, rate, payment`*`)` function to return the number of years that it will take to pay off the loan. Use it to write a program that asks the user for the amount owed, annual interest rate, and monthly payment amount, and then prints the number of years it will take to pay.

    (b) If the monthly payment is not greater than the amount of interest for the first month, the loan will never be paid. Add a test to report this to the user instead of calling the function.

11. Write a program to repeatedly allow a user to calculate the area of circles. Stop when the input is -1; make sure the user knows that. Use appropriate functions and provide opening and closing messages. The area of a circle is πr^2.

12. Write a program to repeatedly allow a user to calculate the volume of spheres. Stop when the input is -1; make sure the user knows that. Use appropriate functions and provide opening and closing messages. The volume of a sphere is $\frac{4}{3}\pi r^3$.

13. Write a program to repeatedly allow a user to do mile-to-kilometer conversions. Stop when the input is -1; make sure the user knows that. Use appropriate functions and provide opening and closing messages. One mile is approximately 1.609 km.

14. Write a program to repeatedly allow a user to calculate maximum heart rate estimates, using the formula $208 - 0.7y$ for a person y years of age. Stop when the input is -1; make sure the user knows that. Use appropriate functions and provide opening and closing messages.

EXERCISES

1. Evaluate these Python expressions:
   (a) `list(range(6))`
   (b) `list(range(1, 20, 3))`
   (c) `[2*i for i in range(5)]`
   (d) `[n**2 for n in range(8)]`
2. Evaluate these Python expressions:
   (a) `list(range(2, 14))`
   (b) `list(range(10, 101, 10))`
   (c) `[2*i + 1 for i in range(15, 20)]`
   (d) `[m**3 for m in range(1, 5)]`
3. Write Python expressions to create these lists:
   (a) `[1, 2, 3, 4, 5, 6]`
   (b) `[2, 6, 10, 14, ..., 46]`
   (c) `[0, 0, 0, 0, ..., 0]` with fifty entries
   (d) A list of 1000 random integers either 0 or 1

4. Write Python expressions to create these lists:
   (a) `[11, 13, 15, 17, 19, 21, 23, 25]`
   (b) `[72, 69, 66, 63, ..., 3]`
   (c) `[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, ..., 4]` with fifty entries
   (d) A list of 1000 random integers between 1 and 6

7. Write a `mean(`*`items`*`)` function to return the average of the values in the list *items*. Test your function on random data.

8. Write a `product(`*`items`*`)` function to return the product of the values in the list *items*. Test your function on random data.

9. Write a `count(`*`target, items`*`)` function to return the number of times *target* appears in the list *items*. Include a program to test your function.

10. Write a `count_evens(`*`items`*`)` function that returns the number of even integers in the list *items*. Test your function on random data.

11. Write a `count_odds(`*`items`*`)` function that returns the number of odd integers in the list *items*. Test your function on random data.

12. Write a `geometric_mean(`*`items`*`)` function to return the geometric mean of the values in the list *items*. The **geometric mean** of x_1, x_2, ..., x_n is
$$(x_1 x_2 \ldots x_n)^{\frac{1}{n}}.$$
Test your function on random data.

13. Write a `sum_of_squares(`*`items`*`)` function to return the sum of the squares of the values in the list *items*. Test your function on random data.

14. Write a `rms(`*`items`*`)` function to return the root mean square of the values in the list *items*. The **root mean square** or **quadratic mean** of x_1, x_2, ..., x_n is
$$\sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}.$$
Test your function on random data.

15. Write a `variance(`*`items`*`)` function to return the variance of the population in the list *items*. The **population variance** of x_1, x_2, ..., x_n is
$$\frac{(x_1 - m)^2 + (x_2 - m)^2 + \cdots + (x_n - m)^2}{n}$$
where m is the mean of x_1, x_2, ..., x_n. Test your function on random data.

16. Write a `std_dev(items)` function to return the standard deviation of a population in the list *items*. The **standard deviation** of a population is the square root of its population variance; see the previous exercise. Test your function on random data.

17. Write a `sample_variance(items)` function to return the sample variance of the list *items*. The **sample variance** of x_1, x_2, …, x_n is

$$\frac{(x_1 - m)^2 + (x_2 - m)^2 + \cdots + (x_n - m)^2}{n - 1}$$

    where m is the mean of x_1, x_2, …, x_n. Test your function on random data.

18. Write a `sample_std_dev(items)` function to return the standard deviation of a sample in the list *items*. The **sample standard deviation** is the square root of its sample variance; see the previous exercise. Test your function on random data.

19. Write a function `randints(a, b, n)` to return a list of n random integers between a and b (inclusive). Use your function to print a list of fifty random integers between 1 and 10.

20. Write a function `randfloats(a, b, n)` to return a list of n random floating-point values between a and b (inclusive). Use your function to print a list of fifty random floats between 0 and 1.

EXERCISES

1. Suppose `nums = [35, 26, 19, 23, 6, 8, 18, 1, 34]`. Give the value of each of these expressions:

   (a) `nums[3]`
   (b) `nums[-1]`
   (c) `nums[:4]`
   (d) `nums[6:]`
   (e) `nums[2:5]`
   (f) `nums[3:7]`
   (g) `nums[1::2]`
   (h) `nums[::-1]`

2. Suppose `nums = [32, 18, 25, 34, 23, 24, 40, 37, 4, 31]`. Give the value of each of these expressions:

   (a) `nums[2]`
   (b) `nums[-1]`
   (c) `nums[:-1]`
   (d) `nums[3:]`
   (e) `nums[1:4]`
   (f) `nums[5:8]`
   (g) `nums[::2]`
   (h) `nums[::-1]`

3. Suppose `items` is a nonempty list. Write Python expressions for:

   (a) The first element (counting from the beginning)
   (b) The last element

4. Suppose `items` is a list with at least three elements. Write Python expressions for:

   (a) The third element (counting from the beginning)
   (b) The next-to-last element

5. Let `items = [7, 4, 27, 26, 2, 38, 19, 10, 34]`. Write expressions using `items` to produce each of these values:

   (a) `38`
   (b) `[7, 4]`
   (c) `[2, 38, 19]`
   (d) `[4, 26, 38, 10]`

6. Let `items = [11, 29, 22, 36, 40, 25, 14, 7, 16]`. Write expressions using `items` to produce each of these values:

   (a) `[29]`

   (b) `[14, 7, 16]`

   (c) `[36, 40, 25, 14]`

   (d) `[16, 7, 14, 25, 40]`

7. Suppose `nums = [49, 41, 3, 46, 9, 12, 20]`. Show the result of each of these assignments, starting over with the original value of `nums` for each.

   (a) `nums[3] = 6`

   (b) `nums[0] += 10`

   (c) `nums[:2] = [0, 1, 2]`

8. Suppose `nums = [16, 40, 37, 32, 29, 47, 26, 50]`. Show the result of each of these assignments, starting over with the original value of `nums` for each.

   (a) `nums[-1] = 10`

   (b) `nums[2] += nums[1]`

   (c) `nums[1:4] = []`

9. Let `items = [34, 35, 13, 2, 29, 38, 7, 38, 5, 22]`. Write one assignment statement (each) that would change `items` to:

   (a) `[34, 35, 13, 2, 29, 38, 38, 38, 5, 22]`

   (b) `[34, 35, 36, 37, 38, 7, 38, 5, 22]`

   (c) `[34, 35, 13, 2, 29, 38, 7, 38, 39, 40, 41]`

10. Let `items = [5, 23, 17, 36, 3, 20, 31, 23, 2, 36]`. Write one assignment statement (each) that would change `items` to:

    (a) `[5, 23, 0, 36, 3, 20, 31, 23, 2, 36]`

    (b) `[5, 23, 17, 36, 36]`

    (c) `[5, 23, 17, 36, 3, 4, 5, 6, 7, 20, 31, 23, 2, 36]`

12. Write a `mymax(`*`items`*`)` function that returns the largest item in the list *items* without using the built-in **`max()`** function. Test your function on a random list of numbers and compare it against **`max()`**.

13. Write a `first(`*`items`*`)` function to return the first item in the list *items*. Include a program to test your function.

14. Write a `last(`*`items`*`)` function to return the last item in the list *items*. Include a program to test your function.

15. Write a `rest(items)` function to return a list containing all but the first item in the list *items*. Include a program to test your function.

16. Write a function `myreversed(items)` to return a list containing the items in *items* in reverse order, meaning opposite of their original order. Do not use the built-in **`reversed()`** function or modify the original list. Test your function on random lists.

18. Write a `median(items)` function to return the median of the values in the list *items*. To find the **median**, sort the values in increasing order, and then if the list has odd length, the median is the middle element; otherwise, if it has even length, the median is the average of the two middle values. Do not modify the original list, and test your function on random lists.

19. Write a `swap(items, i, j)` function to swap `items[i]` with `items[j]`. Write a program to test your function.

20. (Requires Exercise 19.) Write a function `myshuffle(items)` that shuffles the list *items* without using the `shuffle()` function from the `random` module. Use the `swap()` function from the previous exercise and this outline:

    ```
    for every index i:
      pick a random index $j \geq i$
      swap items i and j
    ```

    Include a program to test your function.

EXERCISES

1. Suppose `a = [3, 15, 30, 19, 5, 10]` and `b = [32, 39, 35, 27]`. Determine the value of each of these expressions:
   (a) `a + b`
   (b) `b[1:] + a`
   (c) `3 * b`
2. Suppose `a = [47, 42, 44, 12]` and `b = [40, 38, 2, 32, 3]`. Determine the value of each of these expressions:
   (a) `b + 2*a`
   (b) `a[:3] + b[2:]`
   (c) `a[3] + b[2]`
3. Suppose `a = [3, 15, 30, 19, 5, 10]` and `b = [32, 39, 35, 27]`.
   (a) Determine the effect of `a += b`.
   (b) Determine the effect of `a = [b[-1]] + a`.
4. Suppose `a = [47, 42, 44, 12]` and `b = [40, 38, 2, 32, 3]`.
   (a) Determine the effect of `b = a[:2] + b`.
   (b) Determine the effect of `b += b`.

5. Suppose `a` and `b` are lists.
   (a) Write an expression to concatenate the first three elements of `a` with the last four elements of `b`.
   (b) Write an expression to concatenate `a` with the first element of `b`.
6. Suppose `a` and `b` are lists.
   (a) Write an expression to concatenate all but the first element of `a` with the first two elements of `b`.
   (b) Write an expression to append the last element of `a` onto `b`.
7. Write a Python statement to set `counts` to name a list of 25 0's.
8. Write a Python statement to append 15 0's to the end of the list `counts`.
9. Determine the output of this code:

```
result = []
for i in range(4):
    result += [2*i + 1]
    print(i, result)
```

10. Determine the output of this code:

```
result = []
for i in range(5):
    print(i, result)
    result += [25 - 3*i]
```

11. Determine the value of: `[i**2 for i in range(10) if i%2 == 1]`
12. Determine the value of: `[2**i for i in range(10) if i%3 == 1]`

15. Write an `odds(`*`items`*`)` function to return a list of only the odd integers from the list *items*. Test your function on random lists.
16. Write a function `myreversed(`*`items`*`)` to return a list containing the items in *items* in reverse order using a list accumulator. Do not use the built-in **`reversed()`** function or modify the original list. Test your function on random lists.

17. Write a function `rotate_right(`*`items`*`)` to return a list containing the elements in *items* shifted one place to the right, wrapping the last element around to the front. For example,

$$\texttt{[0, 1, 2, 3]} \xrightarrow{\text{rotate right}} \texttt{[3, 0, 1, 2]}$$

Do not modify the original list. Include a program to test your function.

18. Write a function `rotate_left(`*`items`*`)` to return a list containing the elements in *items* shifted one place to the left, wrapping the first element around to the end of the list. For example,

$$\texttt{[0, 1, 2, 3]} \xrightarrow{\text{rotate left}} \texttt{[1, 2, 3, 0]}$$

Do not modify the original list. Include a program to test your function.

19. Write a function `randints(a, `*`b`*`, `*`n`*`)` to return a list of n random integers between a and b (inclusive) using a list accumulator. Use your function to print a list of fifty random integers between 1 and 10.

20. Write a function `randfloats(a, `*`b`*`, `*`n`*`)` to return a list of n random floating-point values between a and b (inclusive) using a list accumulator. Use your function to print a list of fifty random floats between 0 and 1.

21. Write a function `divisors(`*`n`*`)` to return a list of the proper positive divisors of the integer n, including 1 but excluding n itself. Use your function to print a table of values of the divisors of 2 through 20.

22. (Requires Exercise 21.) Write a function `isperfect(`*`n`*`)` to return `True` if n is a **perfect** number, meaning it is positive and equal to the sum of its proper divisors. Use the previous exercise, and write a program to find all perfect numbers less than 10,000.

EXERCISES

7. Write a `contains(`*`target, items`*`)` function to return `True` if the list *items* contains *target* and `False` otherwise. Do not use **`in`** or **`not in`**. Include a program to test your function.

8. Write a `find_last(`*`target, items`*`)` function to return the index of the last occurrence of *target* in *items*, or -1 if it is not found. Include a program to test your function.

9. Write a program that uses binary search to guess a number between 1 and 100 chosen by the user. After each guess, the user indicates if the guess was too high, too low, or correct. You do not need to write any functions other than `main()`.

10. Write a `binary_search(`*`target`*`, `*`items`*`)` function that uses binary search to return the index where *target* is located in the sorted list *items* or -1 if the item is not found. Include a program to test your function. Do not worry about duplicate items.

11. You might have expected "`+`" to arithmetically add the elements of two lists, as long as they were lists of numbers. Write an `sumlists(`*`a`*`, `*`b`*`)` function to return a list of the sums of corresponding items in *a* and *b* (`a[0] + b[0]`, etc.). If one list is shorter, treat missing items as 0. Test your function on random data.

12. Write a `fibonacci(`*`n`*`)` function to return a list of the first n Fibonacci numbers. The **Fibonacci numbers** start with two 1's and then each next element is the sum of the two previous values:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Include a program to test your function.

EXERCISES

1. Trace the computation of add$(6, 3)$ using this recursive definition of addition for integer $n \geq 0$:

$$\text{add}(m, n) = \begin{cases} m & \text{if } n = 0 \\ 1 + \text{add}(m, n-1) & \text{if } n > 0 \end{cases}$$

2. Write a recursive Python function `add(m, n)` to compute the add() function in the previous exercise. Test your function against regular addition.

3. Trace the computation of mul$(5, 4)$ using this recursive definition of multiplication for integer $n \geq 0$:

$$\text{mul}(m, n) = \begin{cases} 0 & \text{if } n = 0 \\ m + \text{mul}(m, n-1) & \text{if } n > 0 \end{cases}$$

4. Write a recursive Python function `mul(m, n)` to compute the mul() function in the previous exercise. Test your function against regular multiplication.

5. Trace the computation of power$(2, 5)$ using this recursive definition of exponentiation for $m > 0$, integer $n \geq 0$:

$$\text{power}(m, n) = \begin{cases} 1 & \text{if } n = 0 \\ m \cdot \text{power}(m, n-1) & \text{if } n > 0 \end{cases}$$

6. Write a recursive Python function `power(m, n)` to compute the power() function in the previous exercise. Test your function against the built-in `pow()` function.

7. Write a recursive Python function `fib(n)` to return the nth **Fibonacci number**, where each term is the sum of the two previous values:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

For example, `fib(1) = 1` and `fib(4) = 3`. Include a program to test your function.

EXERCISES

1. Suppose `word = "albatross"`. Give the value of each of these expressions:

   (a) `word[3]`
   (b) `word[:4]`
   (c) `word[2:5]`
   (d) `word[::2]`

2. Suppose `word = "palatable"`. Give the value of each of these expressions:

   (a) `word[-1]`
   (b) `word[4:]`
   (c) `word[1:6:2]`
   (d) `word[::-1]`

3. Suppose `word = "rehearse"`. Give the value of each of these expressions:

   (a) `word[5]`
   (b) `word[:-1]`
   (c) `word[3:6]`
   (d) `word[1::2]`

4. Suppose `word = "shellfish"`. Give the value of each of these expressions:

   (a) `word[-1]`
   (b) `word[5:]`
   (c) `word[2:5]`
   (d) `word[::-1]`

5. Determine the value of these expressions:

   (a) `"event" in "seventies"`
   (b) `"seed" in "squelched"`

6. Determine the value of these expressions:

   (a) `"dome" in "seldom errs"`
   (b) `"pig" in "pigeonhole"`

7. Let `phrase = "string quartet in d minor"`. Write expressions using `phrase` to produce each of these values:

   (a) `"string"` (b) `"quart"` (c) `"minor"` (d) `"ring"`

8. Let `phrase = "delivery service"`. Write expressions using `phrase` to produce each of these values:

   (a) `"deli"` (b) `"very"` (c) `"ice"` (d) `"reviled"`

9. Let `phrase = "twenty-four hours"`. Write expressions using `phrase` to produce each of these values:

   (a) `"twenty"` (b) `"four"` (c) `"hour"` (d) `"newt"`

10. Let `phrase = "salami sandwich"`. Write expressions using `phrase` to produce each of these values:

    (a) `"sand"` (b) `"ami"` (c) `"ich"` (d) `"alas"`

11. Write a boolean expression that is `True` if the first character of the string `word` is a vowel (not including "y").

17. Write a conjugation function for a language and regular verb family of your choice. Include a program to test your function.

18. Write a `username(first, last)` function to return a computer system username given a person's first and last names. The username is made up of at most 12 characters of the last name, followed by the first initial of the first name, followed by a "1." Include a program to test your function.

19. Write a `pig_latin(`*word*`)` function to translate *word* into Pig Latin, assuming *word* either starts with a vowel or only one consonant. Include a program to test your function.

20. Write a `pig_latin(`*word*`)` function to translate *word* into Pig Latin without the limitation of the previous exercise. Include a program to test your function.

21. Write an `adverb(adjective)` function to return the adverb form of *adjective* in English. Do not worry about irregularities, but try to capture some of the regular forms. Include a program to test your function.

22. Write a `plural(noun)` function to return the plural form of *noun* in English. Do not worry about irregularities, but try to capture some of the regular forms. Include a program to test your function.

23. Like lists, Python strings are naturally recursive: they can get gradually smaller with a slice that removes one character, and the base case is an empty string. Write a recursive `length(s)` function to return the length of the string *s* without using the built-in **`len()`** function. Test your function against **`len()`**.

EXERCISES

3. Determine the output of this code:

```
result = ""
for i in range(1, 5):
    result += str(i**2)
    print(i, result)
```

4. Determine the output of this code:

   ```
   result = ""
   for c in "apple":
       print(c, result)
       result = c + result
   ```

5. Write a `reverse(s)` function to return a copy of the string *s* in reverse order using a slice. Include a program to test your function.

6. Write a `reverse(s)` function to return a copy of the string *s* in reverse order using an accumulator rather than a slice. Include a program to test your function.

8. Write a `random_rna(length)` function to return a random fragment of mRNA of the given length. Messenger RNA, or **mRNA** is made of the same bases as DNA, except that uracil (U) replaces thymine (T). Include a default length and a program to test your function.

9. Use a string loop to write a function `is_dna(s)` to return `True` if the string *s* consists entirely of DNA bases and otherwise return `False`. Test your function on random strings of DNA, as well as other, non-DNA strings.

10. Write a function `is_rna(s)` to return `True` if the string *s* consists only of mRNA bases and otherwise return `False`. Test your function on random strings of mRNA and DNA.

11. A fragment of DNA is a **palindrome** if it is the same as its reverse complement. (This is not the same as a word palindrome.) Write a function `is_palindrome(dna)` to return `True` if the given *dna* is palindromic, and otherwise return `False`. Use your function to write a program that finds a palindrome of length 10 by testing randomly generated strings of DNA until it finds one.

12. DNA **transcription** can be viewed as producing a strand of mRNA that is the same as the non-template (coding or sense) strand except every thymine (T) is replaced by uracil (U). Write a function `transcription(sense_dna)` to return the corresponding mRNA for *sense_dna*. Test your function on random strings of DNA.

    $\longrightarrow$ Note: Transcription actually works on the template (antisense) strand.

13. Write a `vowels(s)` function to return a string consisting only of the vowels in the string *s*, in the order they appear. Include a program to test your function.

14. Write a `hide_vowels(s)` function to return a copy of the string *s* with each vowel replaced by a hyphen "`-`." Include a program to test your function.

15. Write a `random_bits(length)` function to return a string of random bits (0's and 1's) of the given *length*. Include a default value for *length* and a program to test your function.

16. Write a `random_digits(length)` function to return a string of random digits (0 through 9) of the given *length*. Include a default value for *length* and a program to test your function.

17. Write a `random_hex(length)` function to return a string of random hexadecimal digits (0–9, A–F) of the given *length*. Include a default value for *length* and a program to test your function.

18. Write a recursive `reverse(s)` function to return a copy of the string *s* in reverse order. Include a program to test your function.

19. Write a `romanchar_to_int(c)` function to return the integer value of any single-character Roman numeral *c*: I, V, X, L, C, D, or M. Return 0 for any other character. Include a program to test your function.

20. (Requires Exercise 19.) Write a `romanpair_to_int(pair)` function to return the integer value of any valid subtractive pair of Roman numerals, such as *pair* = `"IX"`. Include a program to test your function.

21. (Requires Exercises 19 and 20.) Write a `roman_to_int(roman)` function to return the integer value of the Roman numeral given in the string *roman*. Include a program to test your function.

22. (Requires Exercise 21.) Write a `largest_roman(n)` function to return the highest-valued single Roman numeral or subtractive pair with value $\leq n$. Include a program to test your function.

23. (Requires Exercises 21 and 22.) Write an `int_to_roman(n)` function to return a string representing the Roman numeral of n, for any $n > 0$. Include a program to test your function on random integers.

24. (Requires Exercise 23.) Write a `random_roman()` function to return a random valid Roman numeral. Include a program to test your function.

25. The last digit in a 13-digit ISBN book identification number is a **check digit** to help catch transmission errors. Research how to compute the check digit, and then write a `check13(code)` function to return the check digit for the given 12-digit string *code*. Include a program to test your function.

22. Write a function `transcription(`*`sense_dna`*`)` to return the corresponding mRNA for *sense_dna* (see page 104). Allow *sense_dna* to be upper, lower, or mixed case. Test your function on random strings of mixed-case DNA.

23. Write a function `is_ly(`*`s`*`)` to return `True` if *s* is a single word (no spaces) ending in "-ly," like many adverbs. Include a program to test your function.

24. Write a function `is_identifier(`*`s`*`)` to return `True` if *s* consists entirely of alphabetic characters, underscores (`_`), and, except for the first character, the digits 0–9; otherwise, it returns `False`. Include a program to test your function.

25. Write a function `is_palindrome(`*`s`*`)` to return `True` if *s* is a **palindrome**, that is, a phrase that reads the same backward as forward, ignoring any punctuation or whitespace, as well as whether characters are upper- or lowercase. For example, "Madam, I'm Adam!" should return `True`. Include a program to test your function.

26. Write `encode()` and `decode()` functions for the Caesar cipher of Exercise 24 on page 110 that maintain upper- and lowercase letters, as well as spaces and punctuation. Include a program to test your functions.

EXERCISES

1. Which of the following types may be used as keys in a Python dictionary: integer, float, string, list, boolean, dictionary? Explain your answers.

2. Which of the following types may be used as values in a Python dictionary: integer, float, string, list, boolean, dictionary? Explain your answers.

3. Write a Python statement to create a dictionary named `rating`, where each key is a movie title and the value is your rating of that film from 1 (low) to 5 (high). Include at least four entries.

4. Write a Python statement to create a dictionary named `friend`, where each key is a person's name and the value is a list of their friends' names. Include at least four entries.

5. Using the `population` dictionary on page 136, do the following:

   (a) Write statements to add three new entries (without re-creating the dictionary).

   (b) Write a statement to delete the entry for New York.

   (c) Write an expression to test to see if there is an entry for Tokyo.

   (d) Give the value of: `list(population.keys())`.

6. Using the `status` dictionary on page 136, do the following:

   (a) Write statements to add three new entries (without re-creating the dictionary).

   (b) Write a statement to change the entry for status 200 to "OK."

   (c) Write an expression that gives the number of entries in `status`.

   (d) Give the value and explain the effect of: `status.pop(404)`.

10. Write a function `letter_frequency(`*`text`*`)` to return the frequencies of the letters (only) in the string *text*, ignoring upper- and lowercase. Include a program to test your function. Letter frequencies are important for encryption, compression, modern keyboard designs, and some games.

11. Write a function `char_frequency(`*`text`*`)` to return the frequencies of every character in the string *text*. Include a program to test your function.

12. Write a function `forms(`*`verb`*`)` to return the present indicative conjugated forms of *verb* for any regular -ar, -er, or -ir Spanish verb. Use a Python dictionary to store the list of endings for each family of verbs, and include a program to test your function.

14. Write a function `top(`*`n, items`*`)` to return a list of the top *n* elements in the list *items*, based on frequency. Include a program to test your function.

15. Write a `mode(`*`items`*`)` function to return the mode of the values in the list *items*. The **mode** of a data set is the value that occurs most often; if there is a tie, return any one of the items with the highest count. Include a program to test your function.

16. Write a program to find the word(s) with the most anagrams from a text dictionary file. Hint: use a Python dictionary to count how many words have the same string of sorted characters.

17. Write a program to provide an interactive dictionary, where a user can add, change, and look up definitions. The program starts with an empty dictionary, and then entries are gradually created by the user. The start of an interactive session might look like this, with the user's input underlined:

```
Welcome to PyDict
        [a]dd or change a definition
        [l]ookup a word
        [q]uit
Your choice: l
Word to look up: python
Not in dictionary
        [a]dd or change a definition
        [l]ookup a word
        [q]uit
Your choice: a
Word: python
Definition: A long, large snake.
        [a]dd or change a definition
        [l]ookup a word
        [q]uit
Your choice: l
Word to look up: python
Definition: A long, large snake.
        [a]dd or change a definition
        [l]ookup a word
        [q]uit
```

3. Write a function `respond(`*`text`*`)` to return a more appropriate response based on what the user has said in *text*. Set up a keyword-response dictionary so that if the user types one of the keywords within *text*, a more targeted response can be given. Return a random response if none of the keywords match.

4. Ask the user for his or her name, and incorporate their name into some of the program's responses, both random and keyword based.

5. Make adjustments so that important terms such as keywords or "Bye" are found regardless of punctuation or upper-/lowercase.