COMP 1023 Introduction to Python Programming

Exercises on Looping

COMP 1023 Teaching Team

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

Exercise 1

Write a program that reads an unspecified number of integers, determines how many even and odd values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input 0. Display the average as a floating-point number.

```
Enter an integer, the input ends if it is 0: 8
Enter an integer, the input ends if it is 0: 3
Enter an integer, the input ends if it is 0: -4
Enter an integer, the input ends if it is 0: 9
Enter an integer, the input ends if it is 0: 7
Enter an integer, the input ends if it is 0: 5
Enter an integer, the input ends if it is 0: 0
The number of evens is 2
The number of odds is 4
The total is 28
The average is 4.666666666666667
```

Exercise 1 Solutions

```
even_count = odd_count = total = count = 0
while True:
    num = int(input("Enter an integer, the input ends if it is 0: "))
    if num == 0:
        break
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1
    total += num
    count += 1

if count > 0:
    average = total / count
    print(f"The number of evens is {even_count}")
    print(f"The number of odds is {odd_count}")
    print(f"The total is {total}")
    print(f"The average is {average}")
else:
    print("No numbers were entered except 0")
```

Exercise 2

Write a program that displays, five numbers per line, all the numbers from 1,000 to 5,000 that are divisible by 11 and 17.

Exercise 2 Solutions

```
count = 0
for num in range(1000, 5001):
    if num % 11 == 0 and num % 17 == 0:
        print(num, end=' ')
        count += 1
        if count % 5 == 0:
            print()
```

Exercise 3

Use a while loop to find the largest integer n such that $n^3 - n^2$ does not exceed 1,000.

Exercise 3 Solutions

```
n = 1
while n**3 - n**2 <= 1000:
    n += 1

print(f"The largest integer n is {n-1}")
print(f"n^3 - n^2 = { (n-1)**3 - (n-1)**2 } <= 1000")
print(f"For n = {n}, n^3 - n^2 = {n**3 - n**2} > 1000")
```

Exercise 4

Find the greatest common divisor of two integers n1 and n2 by doing the following: First find d to be the minimum of n1 and n2, then check whether d, d-1, d-2, ..., 2, or 1 is a divisor for both n1 and n2 in this order. The first such common divisor is the greater common divisor for n1 and n2.

Exercise 4 Solutions

```python
# Get input from user
n1 = int(input("Enter first integer: "))
n2 = int(input("Enter second integer: "))

# Find the minimum of n1 and n2
d = min(n1, n2)

# Check divisors from d down to 1
while d > 0:
    if n1 % d == 0 and n2 % d == 0:
        print(f"The greatest common divisor is {d}")
        break
    d -= 1
```

Exercise 5

Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid, as shown in the following sample run:

```
Enter the number of lines: 7
          1
        2 1 2
      3 2 1 2 3
    4 3 2 1 2 3 4
  5 4 3 2 1 2 3 4 5
6 5 4 3 2 1 2 3 4 5 6
7 6 5 4 3 2 1 2 3 4 5 6 7
```

Exercise 5 Solutions

```python
n = int(input("Enter the number of lines: "))

for i in range(1, n + 1):
    # Print leading spaces
    print("  " * (n - i), end="")

    # Print descending numbers
    for j in range(i, 0, -1):
        print(j, end=" ")

    # Print ascending numbers
    for j in range(2, i + 1):
        print(j, end=" ")

    print()
```

Exercise 6

Write a nested for loop that displays the following output:

```
                    1
                  1   2   1
                1   2   4   2   1
              1   2   4   8   4   2   1
            1   2   4   8  16   8   4   2 1
          1 2   4   8 16  32 16   8   4 2 1
        1 2 4   8 16 32  64 32 16   8 4 2 1
1 2 4 8 16 32 64 128 64 32 16 8 4 2 1
```

Exercise 6 Solutions

```
rows = 8

for i in range(rows):
    # Print leading spaces
    print(" " * (3 * (rows - i - 1)), end="")

    # Print left side (ascending powers of 2)
    for j in range(i + 1):
        print(2 ** j, end=" " * (3 - len(str(2 ** j))))

    # Print right side (descending powers of 2, skipping 1)
    for j in range(i - 1, -1, -1):
        print(2 ** j, end=" " * (3 - len(str(2 ** j))))

    print()
```

Exercise 7

Write a program to sum the following series:

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \ldots + \frac{95}{97} + \frac{97}{99}$$

Exercise 7 Solutions

```
total = 0
for numerator in range(1, 98, 2):
    denominator = numerator + 2
    total += numerator / denominator

print(f"The sum of the series is: {total}")
```

Exercise 8

You are often asked to compute the mean and standard deviation of data. The mean is simply the average of the numbers. The standard deviation is a statistics that tells you how tightly all the various data are clustered around the mean in a set of data. For example, what is the average age of the students in the class? How close are the ages? If all the students are the same age, the deviation is 0. Write a program that prompts the users to enter ten numbers, and displays the mean and standard deviation of these numbers using the following formula:

$$\texttt{mean} = \frac{1}{n}\sum_{i=1}^{n} x_i, \ \texttt{deviation} = \sqrt{\frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right)}$$

```
Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: 4.5
Enter a number: 5.6
Enter a number: 6
Enter a number: 7
Enter a number: 8
Enter a number: 9
Enter a number: 10
The mean is 5.61 and the standard deviation is 2.997943739743404
```

Exercise 8 Solutions

```python
import math

# Initialize variables
n = 10
sum_x = 0
sum_x_squared = 0

# Read 10 numbers from user
for i in range(n):
    x = float(input("Enter a number: "))
    sum_x += x
    sum_x_squared += x * x

# Calculate mean and standard deviation
mean = sum_x / n
deviation = math.sqrt((sum_x_squared - (sum_x * sum_x) / n) / (n - 1))

# Display results
print(f"The mean is {mean} and the standard deviation is {deviation}")
```

Exercise 9

Write a program that prompts the user to enter a positive integer n and displays the first n numbers in the Fibonacci sequence. The Fibonacci sequence starts with 0 and 1, and each subsequent number is the sum of the two preceding ones. Validate the input to ensure n is a positive integer.

```
Enter the number of Fibonacci terms to generate: 8
Fibonacci sequence:
0 1 1 2 3 5 8 13
```

Exercise 9 Solutions

```
n = int(input("Enter the number of Fibonacci terms to generate: "))

if n <= 0:
    print("Please enter a positive integer.")
else:
    print("Fibonacci sequence:")
    a, b = 0, 1
    for i in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

Exercise 10

Write a program that prompts the user to enter a range (start and end numbers) and displays all prime numbers within that range. A prime number is a number greater than 1 that has no positive divisors other than 1 and itself.

```
Enter the start of the range: 10
Enter the end of the range: 30
Prime numbers between 10 and 30 are:
11 13 17 19 23 29
```

Exercise 10 Solutions

```python
start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))

print(f"Prime numbers between {start} and {end} are:")

for num in range(start, end + 1):
    if num > 1:
        is_prime = True
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            print(num, end=" ")
```

Exercise 11

An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

For example, 153 is an Armstrong number.

- Digits: 1, 5, 3
- Number of digits: 3
- Calculation: $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

Write a program that finds all Armstrong numbers between 1 and 1000.

```
Armstrong numbers between 1 and 1000:
1
153
370
371
407
```

Exercise 11 Solutions

```
print("Armstrong numbers between 1 and 1000:")

for num in range(1, 1001):
    # Convert number to string to get digits
    digits = str(num)
    power = len(digits)

    # Calculate sum of digits raised to power
    sum_of_powers = sum(int(digit) ** power for digit in digits)

    # Check if it's an Armstrong number
    if num == sum_of_powers:
        print(num)
```

Exercise 12

Write a program that finds all perfect numbers between 1 and 10,000. A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding the number itself).

Definition: A number is perfect if the sum of its divisors (excluding itself) equals the number itself.

Examples:

- 6: Divisors are 1, 2, 3 ($1 + 2 + 3 = 6$), Perfect
- 28: Divisors are 1, 2, 4, 7, 14 ($1 + 2 + 4 + 7 + 14 = 28$), Perfect
- 12: Divisors are 1, 2, 3, 4, 6 ($1 + 2 + 3 + 4 + 6 = 16$), Not perfect

```
Perfect numbers between 1 and 10000:
6
28
496
8128
```

Exercise 12 Solutions

```
print("Perfect numbers between 1 and 10000:")

for num in range(1, 10001):
    divisors_sum = 0

    # Find all divisors and sum them
    for i in range(1, num):
        if num % i == 0:
            divisors_sum += i

    # Check if it's a perfect number
    if divisors_sum == num:
        print(num)
```

Exercise 13

Write a program that calculates the digital root of a number. The digital root is the recursive sum of all the digits in a number until a single-digit number is obtained.

Process:

- Start with any number
- Sum all its digits
- If the result has more than one digit, repeat the process
- Continue until you get a single-digit number

Examples:

- $12345 \rightarrow 1 + 2 + 3 + 4 + 5 = 15 \rightarrow 1 + 5 = 6$
- $9876 \rightarrow 9 + 8 + 7 + 6 = 30 \rightarrow 3 + 0 = 3$
- $999 \rightarrow 9 + 9 + 9 = 27 \rightarrow 2 + 7 = 9$

```
Enter a number: 12345
Digital root of 12345 is 6

Enter a number: 9876
Digital root of 9876 is 3
```

Exercise 13 Solutions

```python
# Get input from user
number = int(input("Enter a number: "))
original = number

# Calculate digital root
while number >= 10:
    digit_sum = 0
    temp = number

    # Sum all digits
    while temp > 0:
        digit_sum += temp % 10
        temp //= 10

    number = digit_sum

print(f"Digital root of {original} is {number}")
```