Learning Objectives

By the end of this worksheet, you will:

- Determine the exact number of iterations of loops with a variety of loop counter behaviours.
- Find the asymptotic running time of programs containing loops.

1. Loop variations. Each of the following functions takes as input a non-negative integer and performs at least one loop. For each loop, determine the exact number of iterations that will occur (in terms of the function’s input), and then use this to determine the simplest Theta expression\(^1\) for the running time of each function. You do not need to prove any “\(g \in \Theta(f)\)” statements here.

Note: each loop body runs in \(\Theta(1)\) time in this question. While this won’t always be the case, such examples allow you to focus on just counting loop iterations here.

```python
1. def f1(n: int) -> None:
   2.     i = 0
   3.     while i < n:
   4.         print(i)
   5.     i = i + 5
```

```python
2. def f2(n: int) -> None:
   2.     i = 4
   3.     while i < n:
   4.         print(i)
   5.     i = i + 1
```

\(^1\)By “simplest,” we mean ignoring constants and slower-growth terms. For example, write \(\Theta(n)\) instead of \(\Theta(2n + 0.3)\).
```python
def f3(n: int) -> None:
    
    # Precondition: n > 0.
    i = 0
    while i < n:
        print(i)
        i = i + (n / 10)
```

```python
def f4(n: int) -> None:
    i = 20
    while i < n * n:
        print(i)
        i = i + 3
```

```python
def f5(n: int) -> None:
    i = 20
    while i < n * n:
        print(i)
        i = i + 3

    j = 0
    while j < n:
        print(j)
        j = j + 0.01
```
2. **Multiplicative increments.** Consider the following function:

```python
def f(n: int) -> None:
    """Precondition: n > 0."""
    i = 1
    while i < n:
        print(i)
        i = i * 2
```

Even though this looks similar to previous examples, the fact that the loop variable \( i \) changes by a multiplicative rather than additive factor requires a more principled approach in determining the number of loop iterations.

(a) Let \( i_0 \) be the value of \( i \) when 0 loop iterations have occurred, \( i_1 \) be the value of \( i \) immediately after 1 loop iteration has occurred, and in general \( i_k \) be the value of \( i \) immediately after \( k \) loop iterations have occurred. For example, \( i_0 = 1 \) (the initial value of \( i \)), \( i_1 = 2 \), and \( i_2 = 4 \). Determine the values of \( i_3, i_4 \), and a general formula for \( i_k \).  

(b) Use your formula from part (a) to determine the exact number of loop iterations that occur, in terms of \( n \). **HINT:** Find the *smallest* value of \( k \) that makes the loop condition false.

(c) Determine the Theta running time for the function \( f \).

(d) Why did we not initialize \( i = 0 \) in this function?

---

2 Of course, if \( n \) is small then not a lot of loop iterations occur. You can think of \( i_k \) as representing the value of \( i \) after \( k \) loop iterations, *if* \( k \) iterations occur.
3. **A more unusual increment.** Consider the following function:

```python
def f(n: int) -> None:
    """Precondition: n >= 2."""
    i = 2
    while i < n:
        print(i)
        i = i * i
```

Analyse the running time of this function using the same technique as the previous question.