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 n), 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
def f1(n: int) -> None:
    i = 0
    while i < n:
        print(i)
        i = i + 5
```

Solution

There are $\left\lceil \frac{n}{5} \right\rceil$ loop iterations. Since each iteration takes constant time, the total runtime of this function is $\Theta(n)$.

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

Solution

There are $\max(n-4, 0)$ loop iterations. Since each iteration takes constant time, the total runtime of this function is also $\Theta(n)$.

[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)
```

Solution

There are exactly 10 loop iterations. Since each iteration takes constant time, the total runtime of this function is $\Theta(1)$.

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

Solution

There are $\max\left(\left\lceil \frac{n^2 - 20}{3} \right\rceil, 0\right)$ loop iterations. Since each iteration takes constant time, the total runtime of this function is $\Theta(n^2)$.

```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
```

Solution

The first loop takes $\Theta(n^2)$ time (this is from `f4` above). The second loop takes $100n$ iterations, and since each iteration takes constant time, the second loop takes $\Theta(n)$ time. Since $n \in \mathcal{O}(n^2)$, the total runtime of this function is $\Theta(n^2)$.

(This is a consequence of one version of the "sum" Big-O/Omega/Theta theorem.)

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 variable `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.[2]

> **Solution**
>
> The general formula is $i_k = 2^k$.
> Comment: you can actually prove that this formula holds for all $k \in \mathbb{N}$ using induction!

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

> **Solution**
>
> The loop condition is the expression `i < n`. So we want to find the smallest value of $k \in \mathbb{N}$ such that $i_k \geq n$ (to make the loop condition false). We can do this using a simple calculation to "solve" for k:
>
> $$\begin{aligned} i_k &\geq n \\ 2^k &\geq n && \text{(using the formula from part (a))} \\ k &\geq \log n && \text{(recall: default log base is 2)} \end{aligned}$$
>
> Since k must be an integer, the smallest value it can be is $\lceil \log n \rceil$.* So then the loop runs for $\lceil \log n \rceil$ iterations before it stops.
>
> *More formally, $\forall n \in \mathbb{Z},\ \forall x \in \mathbb{R},\ n \geq x \Rightarrow n \geq \lceil x \rceil$.

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

> **Solution**
>
> Each loop iteration takes 1 step (because its running time doesn't depend on the input size). So the total running time is $\lceil \log n \rceil$, which is $\Theta(\log n)$.

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

> **Solution**
>
> Try tracing through the code for a few iterations to see what would happen!

[2] Of course, if n is small then not a lot of loop iterations occur. More formally, i_k represents 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.

Solution

The hardest part of this question is finding a general formula for i_k, the value of variable `i` after k iterations. This turns out to be $i_k = 2^{2^k}$ (the best way to find this is by computing the first few values of `i` by hand). We leave the rest of the analysis as an exercise.