def collatz (n: int) -> int:
    """ Precondition : n > 0 """

    steps = 0
    x = n
    while x > 1:
        if x % 2 == 0:
            x = x // 2
        else:
            x = 3 * x + 1
        steps += 1
    return steps

Q: Describe the runtime of collatz as a function of n, for n ∈ N^+.

A: Open question: # iterations needed, i.e., will the loop terminate for all n ∈ N^+?

- Study a variant.
```
def f(n: int) -> int:
    steps = 0
    x = n
    while x > 1:
        if x % 2 == 0:
            x = x // 2
            steps += 1
        else:
            x = 2 * x - 2
        steps += 1
    return steps

goal: find a function \( h(n) \) s.t. \( RT_f(n) \in \Theta(h(n)) \)

i.e. \( RT_f(n) \in O(h(n)) \)

and \( RT_f(n) \in \Omega(h(n)) \)

Unwrapping def:

1. Find \( h_1(n) \) s.t. \( RT_f(n) \in O(h_1(n)) \)
   \( \exists c_1, n_1 \in \mathbb{R}^+, \forall n \in \mathbb{N} \), \( n > n_1 \implies RT_f(n) \leq c_1 \cdot h_1(n) \)

2. Find \( h_2(n) \) s.t. \( RT_f(n) \in \Omega(h_2(n)) \)
   \( \exists c_2, n_2 \in \mathbb{R}^+, \forall n \in \mathbb{N} \), \( n > n_2 \implies RT_f(n) \geq c_2 \cdot h_2(n) \)
   If \( h_1(n) = h_2(n) \) then can conclude \( RT_f(n) \in \Theta(h(n)) \)
(but may not be possible)

How to progress? Try a few values:

\[ x \text{ even } \rightarrow x = x/2 \]
\[ x \text{ odd } \rightarrow x = 2x - 2 \]

Try a few arguments \( n \) to \( f(n) \):

<table>
<thead>
<tr>
<th>( n = 10 )</th>
<th>( x )</th>
<th>( n = 13 )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\# iterations: 5

\# iterations: 2

\# iterations: 4

Observation: once \( x \) is a power of 2, we stay even

<table>
<thead>
<tr>
<th>( n = 16 )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

\# iterations = 4 = \log_2(n)

How to make sense:
\[
\log_2(10) = 3 \ldots \quad \log_2(13) = 3 \ldots
\]
\[
\left\lfloor \log_2(10) \right\rfloor = 4 \\ \left\lfloor \log_2(13) \right\rfloor = 4
\]

Think about 2 consecutive executions of while loop

\[
\text{\textcolor{red}{x}} \text{ odd: } \quad x \rightarrow 2x-2 \rightarrow \frac{2x-2}{2} = x-1
\]

\[
\text{\textcolor{green}{x}} \text{ even: } \quad x \rightarrow \frac{x}{2} \rightarrow \frac{\frac{x}{2}}{2} = \frac{x}{2^2}
\]

\[\text{odd: } 2 \left( \frac{x}{2^2} \right) - 2 = x-2\]

After two consecutive iterations of while loop \(x\) goes down by at least 1.

\[
\text{total \# iterations} \leq 2(n-1)
\]

\[
\uparrow \quad \text{stop } x = 1
\]

\[
\text{consec. steps.}
\]

\[
\therefore R_T_f(n) \leq 2(n-1)
\]

and \( R_T_f(n) \in O(n) \)

Now \( R_T_f(n) \in \Omega(\ ?) \)
We observed \( n = \text{power of 2} \)

\# iterations \( \log_2(n) \)

try \( RT_f(n) \in \mathcal{O}(\log_2(n)) \)

need to prove

\( \exists c, n_2 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n > n_2 \Rightarrow RT_f(n) \geq c_2 \log_2(n) \)

How to prove:

- every time we go through while loop
  - \( x \) decreases by at most \( \frac{1}{2} \)
  - so we get to \( x = 1 \) after at least \( k \) iterations

  \( \frac{n}{2^k} = 1 \)

  \( \Rightarrow \) at least \( k = \log_2(n) \) iterations

\( \therefore RT_f(n) \in \mathcal{O}(n) \), \( RT_f(n) \in \mathcal{O}(\log_2(n)) \)

Can we do better? Find an \( h(n) \) s.t. \( RT_f(n) \in \Theta(h) \)

- turns \( O(n) \) not a "tight" bound.
- can show \( RT_f(n) \in \Theta(\log n) \)

Hint: think about 3 consec. loop iterations.

back to tens where param. is a list of size \( n \)
Does a given list contain an even number?

def has_even(numbers: list of int) -> bool:
    for number in numbers:
        if number % 2 == 0:
            return True
    return False

How is this runtime different from before?

RT depends on both length and content of list.

<table>
<thead>
<tr>
<th>examples</th>
<th>numbers</th>
<th># iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3, 3, 3]</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>[3, 3, 4]</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>[3, 4, 3]</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>[4, 3, 3]</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4 different lists of length 3

4 different counts for fixed n

Runtime graph

RT

```
```

```
```
How to describe the behavior of slice as n grows?
One option — focus on extrema (now)
— average (later)

Define:

1. Worst case runtime
   \[ \text{WC}_f(n) = \max \{ \text{runtime of } f(x) | x \text{ has size } n \} \]

2. Best case runtime
   \[ \text{BC}_f(n) = \min \{ \text{runtime of } f(x) | x \text{ has size } n \} \]

Note: max/min values of same set.

Logical stunt for this on hr 2 worksheet.
Recall: Goal: describe the runtime of a computer function

Have seen 3 situations:

1. Code where all loops run to completion
   - easy to get $\Theta$ bound

2. Code where loops sometimes finish earlier

Want to find $h(n)$, $h'(n)$ s.t.
\[ \text{RT}_{f_2}(n) \in O(h(n)) \quad \text{and} \quad \text{RT}_{f_2}(n) \in \Omega(h'(n)) \]

when $h'(n) \in \Theta(h(n))$ get $\text{RT}_{f_2}(n) \in \Theta(h(n))$
Code where runtime depends on size of input \( n \) and value of input

E.g. `def has_even(x):` for \( i \) in \( x \):
    if \( i \% 2 == 0 \):
        return True
    return False

- For each \( n \), \( \text{RT}_n(x) \) is a slice of value has_even
- \( WC_f(n) = \max \{ \text{runtime of } f(x) \mid \text{input } x \text{ has size } n \} \) (same set)
- \( BC_f(n) = \min \{ \text{runtime of } g(f(x)) \mid \text{input } x \text{ has size } n \} \)
- Goal: get bound for \( WC_f(n) \) and also for \( BC_f(n) \)
- Search for \( g(n) \) and \( g'(n) \) s.t.
$\mathcal{W}_f(n) \in O(g(n))$ and $\mathcal{W}_f(n) \in \Omega(g(n))$

define: $g' \in \Theta(g)$ so then $\mathcal{W}_f(n) \in \Theta(g(n))$
(wish)
(and similarly for $\mathcal{B}_f(n)$)

unpack the definitions:

$\mathcal{W}_f(n) \in O(g(n))$

$\iff \exists c_0, n_0 \in \mathbb{R}^+, \forall n \geq n_0 \implies \mathcal{W}_f(n) \leq c_0 g(n)$

$\max \sum \text{runtime of } f(x) \mid \text{input } x \text{ has size } n_x^i \leq c_0 g(n)$

- if $\max \text{ in set } S \leq c_0 g(n)$
  then all values in $S \leq c_0 g(n)$

$\forall \text{ inputs } x \text{ of size } n, \text{ runtime of } f(x) \leq c_0 g(n)$

back to has even function:

```
def f(x):
    for i in x:
        if i % 2 == 0
            return T
        return F
```
The number of loop iterations is at most $n$
1 basic operation for 'return False'

\[ \text{runtime of has even}(n) \leq n + 1 \]

For $n \geq 1$,

\[ \forall \text{ inputs } x \text{ of size } n, \text{ runtime of has even }(x) \leq 2^n \]

\[ \text{wc } (n) \in O(n) \]

\[ \text{has even} \]

Unpacking the $\Omega$ definition:

\[ \text{wc } (n) \in \Omega(\,h(n)\,) \]

\[ \exists c, n, \in R^+, \forall n \in N, n > n, \Rightarrow c \cdot h(n) \leq \text{wc } f(n) \]

\[ c \cdot h(n) \leq \max \{|\text{runtime of } f(x)| \text{ input } x \text{ has size } n\} \]

- do all runtimes $f(x)$ need to be $\geq c \cdot h(n)$
- no
- but if $\max \exists 3 \geq c \cdot h(n)$ then some element in $\{3\}$ must be $\geq c \cdot h(n)$
\text{runtime of } f(x) \geq c_i h(n) \text{ for some}
\text{input of size } n
\text{since this } \leq \max \{ \text{runtime size } n \}
\text{so we need to be able to describe an input for each size } n \text{ that has runtime}
\geq c_i h(n)
\text{back to has-even:}
\text{WC has-even } (n) \in \mathcal{O}(n)
\text{wish to show WC has-even } \in \mathcal{\Omega}(n)
\text{how can we force has-even to take at least } n \text{ basic operations}
\text{make the list contain only odd numbers}
\forall n \in \mathbb{N}, \ n \geq 1 \text{ define } X_n = [13, 13, \ldots, 13]
\text{runtime of has-even } (X_n) \text{ is at least } n \text{ basic operations}
\text{Hence for } n \geq N,$
If an input \( x \) of size \( n \), \( \boxed{1. n \leq \text{runtime of } \text{has even}(x)} \)

\[
\text{WC}_{\text{has even}}(n) \in \Omega(n)
\]

\[
\therefore \text{WC}_{\text{has even}}(n) \in \Theta(n).
\]
A more complicated example:

defn: A string $s$ is a palindrome iff it reads the same forwards as backwards. That is

$$s[i] = s[-1-i]$$

$\forall i \in \text{range}(\text{len}(s))$

'reacecar', 'bob', 'c'

defn: A string $s_1$ is a prefix of string $s_2$ iff

$$s_1[i] = s_2[i]$$

$\forall i \in \text{range}(\text{len}(s_1))$

'cat' is a prefix of 'Catherine'

algorithm: given a non-empty string $s$, return the length of the longest prefix of $s$ that is a palindrome

e.g. 'attack' prefixes palindrome len

'a'  T  1

'at'  F

'att'  F  


def palindrome_prefix(s):
    n = len(s)
    for prefix_len in range(n, 0, -1):
        # check whether s[0:prefix_len]
        # is a palindrome
        is_palindrome = True
        for i in range(prefix_len):
            if s[i] != s[prefix_len - 1 - i]:
                is_palindrome = False
                break
        if is_palindrome:
            return prefix_len

Problem: describe WC

The # of iterations is not more than:

\[ n + n-1 + n-2 + \ldots + 1 = \frac{n(n+1)}{2} \]
\[ \text{lower bound: } \begin{align*} s &= 'aaa \ldots a' \quad r.t. n \\ s &= 'baa \ldots a' \quad r.t. n \end{align*} \]