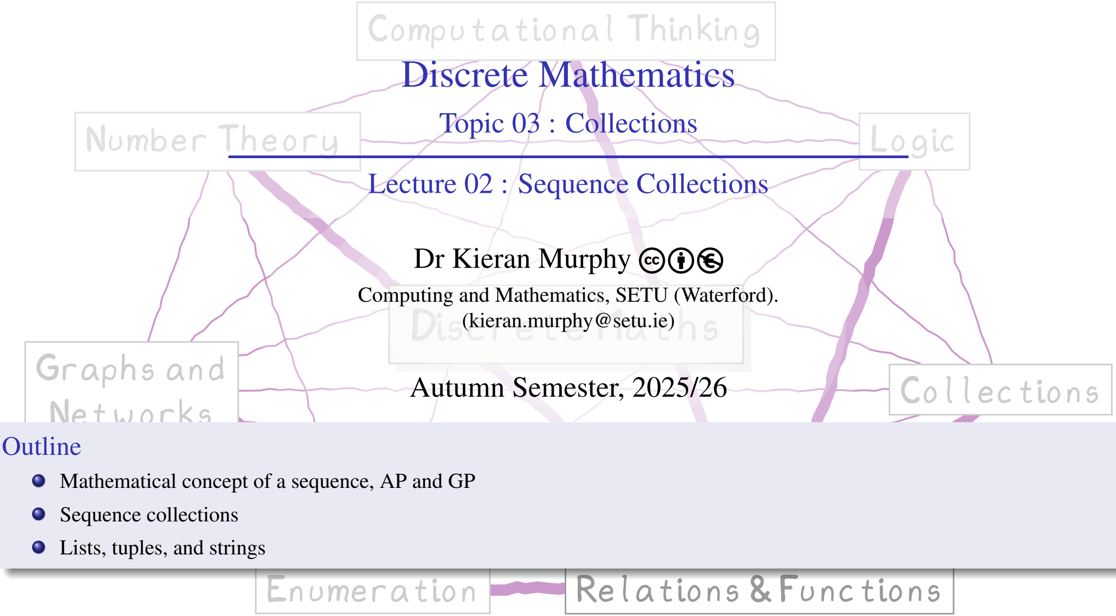


Discrete Mathematics

Topic 03 : Collections

Lecture 02 : Sequence Collections

Dr Kieran Murphy
Computing and Mathematics, SETU (Waterford).
(kieran.murphy@setu.ie)

Autumn Semester, 2025/26

Outline

- Mathematical concept of a sequence, AP and GP
- Sequence collections
- Lists, tuples, and strings

Outline

Sequence

Informally, a sequence is just an ordered list of numbers. Since the order is important we can label the values in the list, starting with zero, then one and so on. This gives us the formal definition of a sequence

Definition 1 (Sequence)

A sequence is a function from the set of natural numbers, $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$ to a some set A. So we have

$$\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & & n & \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \cdots & \downarrow & \cdots \\ a_0 & a_1 & a_2 & a_3 & a_4 & & a_n & \end{array}$$

and

- a_n is the image of n, and is called the nth term/element of the sequence.

- To refer to the *entire* sequence at once, we will write $(a_n)_{n\in\mathbb{N}}$ or $(a_n)_{n\geq 0}$, or if we are being sloppy, just (a_n) (in which case we assume we start the sequence with a_0).
- The numbers in the subscripts are called indices (the plural of index).

Sequence

Informally, a sequence is just an ordered list of numbers. Since the order is important we can label the values in the list, starting with zero, then one and so on. This gives us the formal definition of a sequence

Definition 1 (Sequence)

A sequence is a function from the set of natural numbers, $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$ to a some set A. So we have

$$\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & & n & \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \ldots & \downarrow & \ldots \\ a_0 & a_1 & a_2 & a_3 & a_4 & & a_n & \end{array}$$

and

- a_n is the image of n, and is called the n^{th} term/element of the sequence.

- To refer to the *entire* sequence at once, we will write $(a_n)_{n\in\mathbb{N}}$ or $(a_n)_{n\geq 0}$, or if we are being sloppy, just (a_n) (in which case we assume we start the sequence with a_0).
- The numbers in the subscripts are called indices (the plural of index).

Sequence

Informally, a sequence is just an ordered list of numbers. Since the order is important we can label the values in the list, starting with zero, then one and so on. This gives us the formal definition of a sequence

Definition 1 (Sequence)

A sequence is a function from the set of natural numbers, $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$ to a some set A. So we have

$$\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & & n & \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \cdots & \downarrow & \cdots \\ a_0 & a_1 & a_2 & a_3 & a_4 & & a_n & \end{array}$$

and

- a_n is the image of n, and is called the n^{th} term/element of the sequence.

- To refer to the *entire* sequence at once, we will write $(a_n)_{n\in\mathbb{N}}$ or $(a_n)_{n\geq 0}$, or if we are being sloppy, just (a_n) (in which case we assume we start the sequence with a_0).
- The numbers in the subscripts are called indices (the plural of index).

Examples of Sequences

- The sequence $a_n = n^2$, where $n = 1, 2, 3, \ldots$ has elements

$$1, 4, 9, 16, 25, 36, 49, \ldots$$

- The sequence $a_n = (-1)^n$, where $n = 0, 1, 2, \ldots$ has elements

$$1, -1, 1, -1, 1, -1, \ldots$$

- The sequence $a_n = 2^n$, where $n = 0, 1, 2, \ldots$ has elements

$$1, 2, 4, 8, 16, 32, 64, 128, \ldots$$

- The Fibonacci sequence has elements

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots$$

A Quick Look at Fibonacci Sequence

During 13th century, in Liber Abaci, Fibonacci[*] poses the following question (paraphrasing):

> *Suppose we have two newly-born rabbits, one female and one male. Suppose these rabbits produce another pair of female and male rabbits after one month. These newly-born rabbits will, in turn, also mate after one month, producing another pair, and so on. Rabbits never die. How many pairs of rabbits exist after one year?*

---

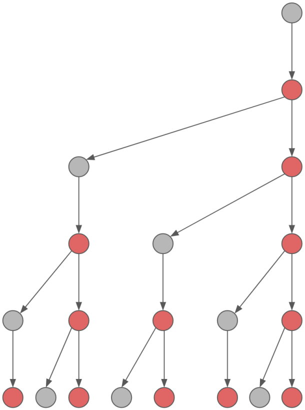
[*] discovered earlier by Indian scholars (Gopãla, before 1135), studying rhythmic patterns

A Quick Look at Fibonacci Sequence

During 13th century, in Liber Abaci, Fibonacci[*] poses the following question (paraphrasing):

> *Suppose we have two newly-born rabbits, one female and one male. Suppose these rabbits produce another pair of female and male rabbits after one month. These newly-born rabbits will, in turn, also mate after one month, producing another pair, and so on. Rabbits never die. How many pairs of rabbits exist after one year?*

Fibonacci's Rabbits



The figure to the right illustrates this process.

- Every point denotes one rabbit pair.
- A grey point denotes a newborn pair (and not ready to reproduce).
- A red point denotes a mature, reproducing pair.

---

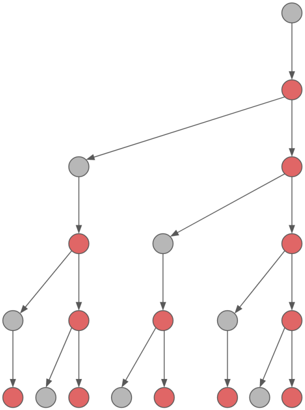
[*] discovered earlier by Indian scholars (Gopãla, before 1135), studying rhythmic patterns

A Quick Look at Fibonacci Sequence

During 13th century, in Liber Abaci, Fibonacci[*] poses the following question (paraphrasing):

> *Suppose we have two newly-born rabbits, one female and one male. Suppose these rabbits produce another pair of female and male rabbits after one month. These newly-born rabbits will, in turn, also mate after one month, producing another pair, and so on. Rabbits never die. How many pairs of rabbits exist after one year?*

Fibonacci's Rabbits



The figure to the right illustrates this process.

- Every point denotes one rabbit pair.
- A grey point denotes a newborn pair (and not ready to reproduce).
- A red point denotes a mature, reproducing pair.

---

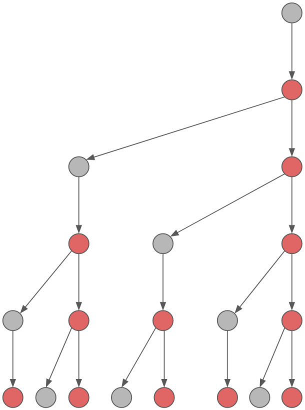
[*] discovered earlier by Indian scholars (Gopãla, before 1135), studying rhythmic patterns

A Quick Look at Fibonacci Sequence

During 13th century, in Liber Abaci, Fibonacci* poses the following question (paraphrasing):

> *Suppose we have two newly-born rabbits, one female and one male. Suppose these rabbits produce another pair of female and male rabbits after one month. These newly-born rabbits will, in turn, also mate after one month, producing another pair, and so on. Rabbits never die. How many pairs of rabbits exist after one year?*

Fibonacci's Rabbits



The figure to the right illustrates this process.

- Every point denotes one rabbit pair.
- A grey point denotes a newborn pair (and not ready to reproduce).
- A red point denotes a mature, reproducing pair.

---

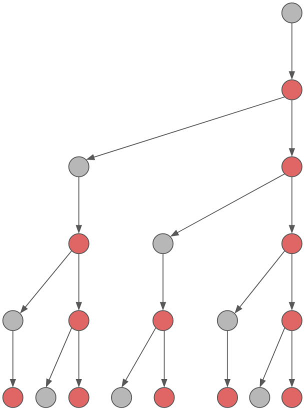
*discovered earlier by Indian scholars (Gopãla, before 1135), studying rhythmic patterns

A Quick Look at Fibonacci Sequence

During 13th century, in Liber Abaci, Fibonacci[*] poses the following question (paraphrasing):

> *Suppose we have two newly-born rabbits, one female and one male. Suppose these rabbits produce another pair of female and male rabbits after one month. These newly-born rabbits will, in turn, also mate after one month, producing another pair, and so on. Rabbits never die. How many pairs of rabbits exist after one year?*

Fibonacci's Rabbits



The figure to the right illustrates this process.

- Every point denotes one rabbit pair.
- A grey point denotes a newborn pair (and not ready to reproduce).
- A red point denotes a mature, reproducing pair.

---

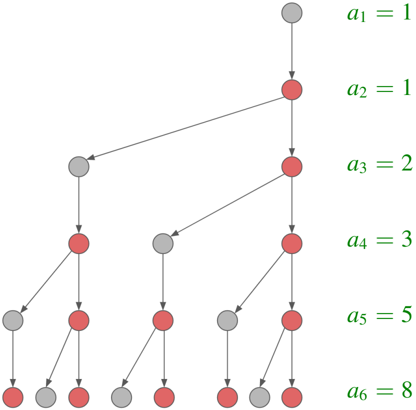
[*] discovered earlier by Indian scholars (Gopãla, before 1135), studying rhythmic patterns

A Quick Look at Fibonacci Sequence

During 13th century, in Liber Abaci, Fibonacci* poses the following question (paraphrasing):

> *Suppose we have two newly-born rabbits, one female and one male. Suppose these rabbits produce another pair of female and male rabbits after one month. These newly-born rabbits will, in turn, also mate after one month, producing another pair, and so on. Rabbits never die. How many pairs of rabbits exist after one year?*

Fibonacci's Rabbits



The figure to the right illustrates this process.

- Every point denotes one rabbit pair.
- A grey point denotes a newborn pair (and not ready to reproduce).
- A red point denotes a mature, reproducing pair.

---

[*] discovered earlier by Indian scholars (Gopãla, before 1135), studying rhythmic patterns

Closed vs Recursive Formula for Sequences

We often need to specify a rule for the general term in the sequence — we have two options:

Definition 2 (Closed Formula and Recursive Definition)

- A closed formula for a sequence a_n is a formula for a_n using a fixed, finite number of operations on n.).
- A recursive definition for a sequence (a_n) consists of a recurrence relation: an equation relating the current term in the sequence, (a_n), to earlier terms in the sequence, (a_{n-1}), (a_{n-2}), $\ldots$(i.e., terms with smaller index) and initial/terminal condition(s).

Example

The Fibonacci sequence $(a_n) = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots)$ has closed formula

$$a_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

and recursive formula

$$\underbrace{a_n = a_{n-1} + a_{n-2}}_{\text{recurrence relation}} \qquad \text{and} \qquad \underbrace{a_0 = 0, \quad a_1 = 1}_{\text{terminal conditions}}$$

Closed vs Recursive Formula for Sequences

We often need to specify a rule for the general term in the sequence — we have two options:

Definition 2 (Closed Formula and Recursive Definition)

- A closed formula for a sequence a_n is a formula for a_n using a fixed, finite number of operations on n.).
- A recursive definition for a sequence (a_n) consists of a recurrence relation: an equation relating the current term in the sequence, (a_n), to earlier terms in the sequence, (a_{n-1}), (a_{n-2}), …(i.e., terms with smaller index) and initial/terminal condition(s).

Example

The Fibonacci sequence $(a_n) = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots)$ has closed formula

$$a_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

and recursive formula

$$\underbrace{a_n = a_{n-1} + a_{n-2}}_{\text{recurrence relation}} \qquad \text{and} \qquad \underbrace{a_0 = 0, \quad a_1 = 1}_{\text{terminal conditions}}$$

Closed vs Recursive Formula for Sequences

We often need to specify a rule for the general term in the sequence — we have two options:

Definition 2 (Closed Formula and Recursive Definition)

- A closed formula for a sequence a_n is a formula for a_n using a fixed, finite number of operations on n.).
- A recursive definition for a sequence (a_n) consists of a recurrence relation: an equation relating the current term in the sequence, (a_n), to earlier terms in the sequence, (a_{n-1}), (a_{n-2}), …(i.e., terms with smaller index) and initial/terminal condition(s).

Example

The Fibonacci sequence $(a_n) = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots)$ has closed formula

$$a_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Hard to obtain, easy to use

and recursive formula

$$\underbrace{a_n = a_{n-1} + a_{n-2}}_{\text{recurrence relation}} \qquad \text{and} \qquad \underbrace{a_0 = 0, \quad a_1 = 1}_{\text{terminal conditions}}$$

Easy to obtain, hard to use

Computing Fibonacci Sequence using Closed Formula

Compute the first 7 terms of the Fibonacci sequence using the closed formula

$$a_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

```python
import math

for n in range(7):

    tmp_1 = (1 + math.sqrt(5)) / 2
    tmp_2 = (1 - math.sqrt(5)) / 2

    a_n = (tmp_1**n - tmp_2**n) / math.sqrt(5)

    print(n, round(a_n))
```

Computing Fibonacci Sequence using Closed Formula

Compute the first 7 terms of the Fibonacci sequence using the closed formula

$$a_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

```python
import math

for n in range(7):

    tmp_1 = (1 + math.sqrt(5)) / 2
    tmp_2 = (1 - math.sqrt(5)) / 2

    a_n = (tmp_1**n - tmp_2**n) / math.sqrt(5)

    print(n, round(a_n))
```

```
0 0
1 1
2 1
3 2
4 3
5 5
6 8
```

Computing Fibonacci Sequence using Recursive Formula

Compute the first 7 terms of the Fibonacci sequence using the recursive formula

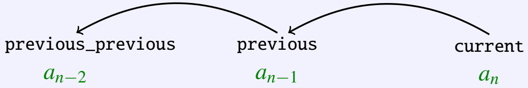
```python
previous_previous = 0
previous = 1

for n in range(7):

    if n == 0:          # terminal condition n=0
        current = 0
    elif n == 1:        # terminal condition n=1
        current = 1
    else:               # recursive formula n>1
        current = previous + previous_previous

        # leapfrog values
        previous_previous = previous
        previous = current

    print(n, current)
```

Computing Fibonacci Sequence using Recursive Formula

Compute the first 7 terms of the Fibonacci sequence using the recursive formula

```
previous_previous = 0
previous = 1

for n in range(7):

    if n == 0:          # terminal condition n=0
        current = 0
    elif n == 1:        # terminal condition n=1
        current = 1
    else:               # recursive formula n>1
        current = previous + previous_previous

        # leapfrog values
        previous_previous = previous
        previous = current

    print(n, current)
```

```
0 0
1 1
2 1
3 2
4 3
5 5
6 8
```



Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned} a_0 &= 3 && \text{(given terminal condition)} \\ a_1 &= 4 && \text{(given terminal condition)} \end{aligned}$$

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned} a_0 &= 3 && \text{(given terminal condition)} \\ a_1 &= 4 && \text{(given terminal condition)} \\ a_2 &= 2 \cdot 4 - 3 = 5 && \text{(use } n = 2 \text{ in recursive formula)} \end{aligned}$$

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned}
a_0 &= 3 && \text{(given terminal condition)}\\
a_1 &= 4 && \text{(given terminal condition)}\\
a_2 &= 2 \cdot 4 - 3 = 5 && \text{(use } n = 2 \text{ in recursive formula)}\\
a_3 &= 2 \cdot 5 - 4 = 6 && \text{(use } n = 3 \text{ in recursive formula)}
\end{aligned}$$

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned}
a_0 &= 3 && \text{(given terminal condition)}\\
a_1 &= 4 && \text{(given terminal condition)}\\
a_2 &= 2 \cdot 4 - 3 = 5 && \text{(use } n = 2 \text{ in recursive formula)}\\
a_3 &= 2 \cdot 5 - 4 = 6 && \text{(use } n = 3 \text{ in recursive formula)}\\
a_4 &= 2 \cdot 6 - 5 = 7 && \text{(use } n = 4 \text{ in recursive formula)}
\end{aligned}$$

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned}
a_0 &= 3 && \text{(given terminal condition)}\\
a_1 &= 4 && \text{(given terminal condition)}\\
a_2 &= 2 \cdot 4 - 3 = 5 && \text{(use } n = 2 \text{ in recursive formula)}\\
a_3 &= 2 \cdot 5 - 4 = 6 && \text{(use } n = 3 \text{ in recursive formula)}\\
a_4 &= 2 \cdot 6 - 5 = 7 && \text{(use } n = 4 \text{ in recursive formula)}\\
a_5 &= 2 \cdot 7 - 6 = 8 && \text{(use } n = 5 \text{ in recursive formula)}
\end{aligned}$$

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned}
a_0 &= 3 && \text{(given terminal condition)}\\
a_1 &= 4 && \text{(given terminal condition)}\\
a_2 &= 2 \cdot 4 - 3 = 5 && \text{(use } n = 2 \text{ in recursive formula)}\\
a_3 &= 2 \cdot 5 - 4 = 6 && \text{(use } n = 3 \text{ in recursive formula)}\\
a_4 &= 2 \cdot 6 - 5 = 7 && \text{(use } n = 4 \text{ in recursive formula)}\\
a_5 &= 2 \cdot 7 - 6 = 8 && \text{(use } n = 5 \text{ in recursive formula)}\\
a_6 &= 2 \cdot 8 - 7 = 9 && \text{(use } n = 6 \text{ in recursive formula)}
\end{aligned}$$

Example

Example 3

Find a_6 in the sequence defined by $a_n = 2a_{n-1} - a_{n-2}$ with $a_0 = 3$ and $a_1 = 4$.

Solution. Using $n = 6$, we know that $a_6 = 2a_5 - a_4$. So to find a_6 we need to find a_5 and a_4. And we repeat this process down to a_0 and a_1. We will use the approach when we define functions.

But for now, we will determine a_6 by starting at a_0 and a_1, and working upwards towards a_6.

$$\begin{aligned}
a_0 &= 3 && \text{(given terminal condition)} \\
a_1 &= 4 && \text{(given terminal condition)} \\
a_2 &= 2 \cdot 4 - 3 = 5 && \text{(use } n = 2 \text{ in recursive formula)} \\
a_3 &= 2 \cdot 5 - 4 = 6 && \text{(use } n = 3 \text{ in recursive formula)} \\
a_4 &= 2 \cdot 6 - 5 = 7 && \text{(use } n = 4 \text{ in recursive formula)} \\
a_5 &= 2 \cdot 7 - 6 = 8 && \text{(use } n = 5 \text{ in recursive formula)} \\
a_6 &= 2 \cdot 8 - 7 = 9 && \text{(use } n = 6 \text{ in recursive formula)}
\end{aligned}$$

Note that in this case a closed formula for a_n exists. Namely, $a_n = n + 3$.
A closed formula is easier to use to calculate a general term, but it is often much harder, if not impossible, to derive.

Computing Sequence using Closed Formula

First 7 terms of the sequence using the closed formula

$$a_n = n + 3$$

```
for n in range(7):

    a_n = n + 3

    print(n, a_n)
```

Computing Sequence using Closed Formula

First 7 terms of the sequence using the closed formula

$$a_n = n + 3$$

```python
for n in range(7):

    a_n = n + 3

    print(n, a_n)
```

```
0 3
1 4
2 5
3 6
4 7
5 8
6 9
```

Computing Sequence using Recursive Formula

Compute the first 7 terms of the Fibonacci sequence using the recursive formula

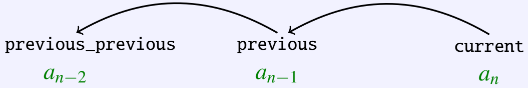
```python
previous_previous = 3
previous = 4

for n in range(7):

    if n == 0:          # terminal condition n=0
        current = 3
    elif n == 1:        # terminal condition n=1
        current = 4
    else:               # recursive formula n>1
        current = 2 * previous - previous_previous

        # leapfrog values
        previous_previous = previous
        previous = current

    print(n, current)
```

Computing Sequence using Recursive Formula

Compute the first 7 terms of the Fibonacci sequence using the recursive formula

```
previous_previous = 3
previous = 4

for n in range(7):

    if n == 0:          # terminal condition n=0
        current = 3
    elif n == 1:        # terminal condition n=1
        current = 4
    else:               # recursive formula n>1
        current = 2 * previous - previous_previous

        # leapfrog values
        previous_previous = previous
        previous = current

    print(n, current)
```

```
0 3
1 4
2 5
3 6
4 7
5 8
6 9
```



previous_previous a_{n-2}

previous a_{n-1}

current a_n

Summation Notation

- The $\sum$ operator is used to denote the addition of elements from a sequence/list.
- It can be implemented using a **`for`** loop in Python/Java/Processing.

Example 4

$$\underbrace{\sum_{k=1}^{10}\left[k^2\right]}_{\substack{\text{“Determine the value of expression within the brackets} \\ \text{as } k=1,2,3,\ldots,10 \text{ and add all the results.”}}} = \underbrace{1^2}_{k=1} + \underbrace{2^2}_{k=2} + \underbrace{3^2}_{k=3} + \underbrace{4^2}_{k=4} + \cdots + \underbrace{10^2}_{k=10}$$

$$= 1 + 4 + 9 + 16 + 25 + 36 + \cdots + 100 = 385$$

```python
result = 0                 # start result with zero - why?
for k in range(1,11):
    term = k*k
    result += term         # shorthand for result = result + term

print(result)
```

```
385
```

Product Notation

- The $\prod$ operator is used to denote the product of elements from a sequence/list.
- It can be implemented using a **for** loop in Python/Java/Processing.

Example 5

$$\underbrace{\prod_{k=1}^{10}\left[k^2\right]}_{\text{“Determine the value of expression within the brackets as } k = 1, 2, 3, \ldots, 10 \text{ and multiply all the results.”}} = \underbrace{1^2}_{k=1} \times \underbrace{2^2}_{k=2} \times \underbrace{3^2}_{k=3} \times \underbrace{4^2}_{k=4} \times \cdots \times \underbrace{10^2}_{k=10}$$

$$= 1 \times 4 \times 9 \times 16 \times 25 \times 36 \times \cdots \times 100 = 13,168,189,440,000$$

```python
result = 1                  # start result with one - why?
for k in range(1,11):
    term = k*k
    result *= term          # shorthand for result = result * term

print(result)
```

```
13168189440000
```

Review Exercises 1 (Sequences) I

Sequences

Question 1:

Expand the following sums

(a) $\sum_{k=4}^{7} k$ **(b)** $\sum_{k=1}^{5} (k^1 - 1)$ **(c)** $\sum_{n=1}^{4} (10^n)$ **(d)** $\sum_{k=1}^{5} (k^1 - 1)$

Question 2:

Write the following expressions using summation notation

(a) $2 + 4 + 6 + 8 + 10$ **(b)** $1 + 4 + 7 + 10$ **(c)** $\frac{1}{4} + \frac{1}{2} + 1 + 2 + 4$

Question 3:

Expand the following sums

(a) $\prod_{k=-4}^{4} k$ **(b)** $\prod_{k=1}^{4} (k^1 - 1)$ **(c)** $\prod_{k \in S} (-1)^k$ where $S = \{2, 4, 6, 7\}$.

Question 4:

For each of the following sequences, determine a recursive definition.

- (a) $2, 4, 6, 10, 16, 26, 42, \ldots.$

Review Exercises 1 (Sequences)

- (b) $5, 6, 11, 17, 28, 45, 73, \ldots$.
- (c) $0, 0, 0, 0, 0, 0, 0, \ldots$.

Question 5:
Show that $a_n = 3 \cdot 2^n + 7 \cdot 5^n$ is a solution to the recurrence relation $a_n = 7a_{n-1} - 10a_{n-2}$. What would the initial conditions need to be for this to be the closed formula for the sequence?

Outline

Arithmetic Progression/Sequence

Definition 6 (Arithmetic Progression/Sequence (AP))

A sequence is called arithmetic if the terms of the sequence differ by a constant.
Suppose the initial term (a_0) of the sequence is a and the common difference is d, then we have sequence

$$a, \qquad a+d, \quad a+2d, \quad a+3d, \qquad \cdots \qquad a+nd, \quad \cdots$$

Arithmetic Progression/Sequence

Definition 6 (Arithmetic Progression/Sequence (AP))

A sequence is called arithmetic if the terms of the sequence differ by a constant.

Suppose the initial term (a_0) of the sequence is a and the common difference is d, then we have sequence

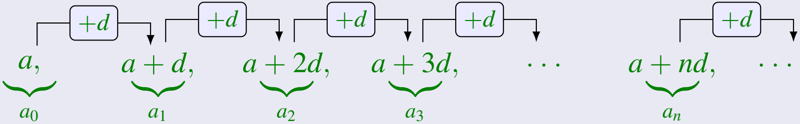
$$\begin{tikzcd}
a, \arrow[r, "+d"] & a+d, \arrow[r, "+d"] & a+2d, \arrow[r, "+d"] & a+3d, \arrow[r, "+d"] & \cdots & a+nd, \arrow[r, "+d"] & \cdots
\end{tikzcd}$$

Arithmetic Progression/Sequence

Definition 6 (Arithmetic Progression/Sequence (AP))

A sequence is called arithmetic if the terms of the sequence differ by a constant.

Suppose the initial term (a_0) of the sequence is a and the common difference is d, then we have sequence



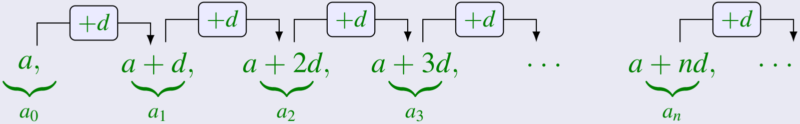
$$\underbrace{a,}_{a_0} \xrightarrow{+d} \underbrace{a+d,}_{a_1} \xrightarrow{+d} \underbrace{a+2d,}_{a_2} \xrightarrow{+d} \underbrace{a+3d,}_{a_3} \xrightarrow{+d} \cdots \qquad \underbrace{a+nd,}_{a_n} \xrightarrow{+d} \cdots$$

Arithmetic Progression/Sequence

Definition 6 (Arithmetic Progression/Sequence (AP))

A sequence is called arithmetic if the terms of the sequence differ by a constant.

Suppose the initial term (a_0) of the sequence is a and the common difference is d, then we have sequence



$$\underbrace{a,}_{a_0} \quad \underbrace{a+d,}_{a_1} \quad \underbrace{a+2d,}_{a_2} \quad \underbrace{a+3d,}_{a_3} \quad \cdots \quad \underbrace{a+nd,}_{a_n} \quad \cdots$$

Recursive definition: $a_n = a_{n-1} + d$ with $a_0 = a$.

Closed formula: $a_n = a + dn$.

Arithmetic Progression/Sequence

Definition 6 (Arithmetic Progression/Sequence (AP))

A sequence is called arithmetic if the terms of the sequence differ by a constant.
Suppose the initial term (a_0) of the sequence is a and the common difference is d, then we have sequence

$$\underbrace{a,}_{a_0} \xrightarrow{+d} \underbrace{a+d,}_{a_1} \xrightarrow{+d} \underbrace{a+2d,}_{a_2} \xrightarrow{+d} \underbrace{a+3d,}_{a_3} \xrightarrow{+d} \cdots \qquad \underbrace{a+nd,}_{a_n} \xrightarrow{+d} \cdots$$

Recursive definition: $a_n = a_{n-1} + d$ with $a_0 = a$.
Closed formula: $a_n = a + dn$.

Example 7

Find recursive definitions and closed formulas for the sequences below. Assume the first term listed is a_0.

- $2, 5, 8, 11, 14, \ldots$.
- $50, 43, 36, 29, \ldots$.

Geometric Progression/Sequence

Definition 8 (Geometric Progression/Sequence (GP))

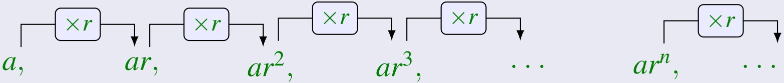
A sequence is called geometric if the ratio between successive terms is constant. Suppose the initial term a_0 is a and the common ratio is r. Then we have, sequence

$$a, \qquad ar, \qquad ar^2, \qquad ar^3, \qquad \cdots \qquad ar^n, \qquad \cdots$$

Geometric Progression/Sequence

Definition 8 (Geometric Progression/Sequence (GP))

A sequence is called geometric if the ratio between successive terms is constant. Suppose the initial term a_0 is a and the common ratio is r. Then we have, sequence

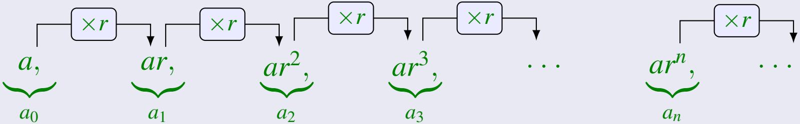


$$a, \quad ar, \quad ar^2, \quad ar^3, \quad \dots \quad ar^n, \quad \dots$$

Geometric Progression/Sequence

Definition 8 (Geometric Progression/Sequence (GP))

A sequence is called geometric if the ratio between successive terms is constant.
Suppose the initial term a_0 is a and the common ratio is r. Then we have, sequence

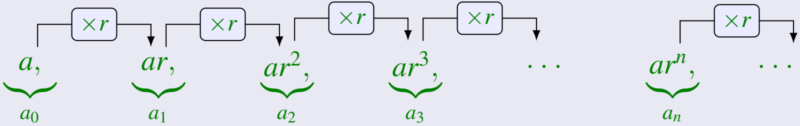


$$\underbrace{a,}_{a_0}\quad \underbrace{ar,}_{a_1}\quad \underbrace{ar^2,}_{a_2}\quad \underbrace{ar^3,}_{a_3}\quad \ldots\quad \underbrace{ar^n,}_{a_n}\quad \ldots$$

Geometric Progression/Sequence

Definition 8 (Geometric Progression/Sequence (GP))

A sequence is called geometric if the ratio between successive terms is constant. Suppose the initial term a_0 is a and the common ratio is r. Then we have, sequence



$$\underbrace{a,}_{a_0}\quad \underbrace{ar,}_{a_1}\quad \underbrace{ar^2,}_{a_2}\quad \underbrace{ar^3,}_{a_3}\quad \ldots\quad \underbrace{ar^n,}_{a_n}\quad \ldots$$

Recursive definition: $a_n = ra_{n-1}$ with $a_0 = a$.
Closed formula: $a_n = ar^n$.

Geometric Progression/Sequence

Definition 8 (Geometric Progression/Sequence (GP))

A sequence is called geometric if the ratio between successive terms is constant.
Suppose the initial term a_0 is a and the common ratio is r. Then we have, sequence

$$\underbrace{a,}_{a_0} \xrightarrow{\times r} \underbrace{ar,}_{a_1} \xrightarrow{\times r} \underbrace{ar^2,}_{a_2} \xrightarrow{\times r} \underbrace{ar^3,}_{a_3} \xrightarrow{\times r} \dots \qquad \underbrace{ar^n,}_{a_n} \xrightarrow{\times r} \dots$$

Recursive definition: $a_n = ra_{n-1}$ with $a_0 = a$.
Closed formula: $a_n = ar^n$.

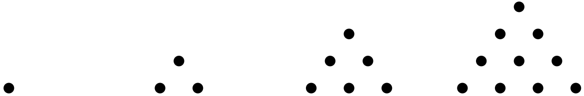
Example 9

Find the recursive and closed formula for the sequences below. Again, the first term listed is a_0.

- $3, 6, 12, 24, 48, \dots$
- $27, 9, 3, 1, 1/3, \dots$

Motivation

Look at the sequence $(T_n)_{n\geq 1}$ which starts $1, 3, 6, 10, 15, \ldots$. These are called the triangular numbers since they represent the number of dots in an equilateral triangle (think of how you arrange 10 bowling pins: a row of 4 plus a row of 3 plus a row of 2 and a row of 1).

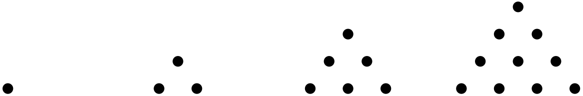


$T_1 = 1$ $T_2 = 3$ $T_3 = 6$ $T_4 = 10$

- Is this sequence arithmetic?
- Is the sequence geometric?

Motivation

Look at the sequence $(T_n)_{n\geq 1}$ which starts $1, 3, 6, 10, 15, \ldots$. These are called the triangular numbers since they represent the number of dots in an equilateral triangle (think of how you arrange 10 bowling pins: a row of 4 plus a row of 3 plus a row of 2 and a row of 1).

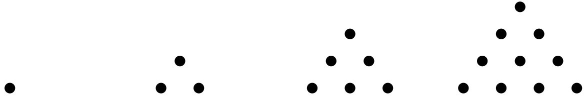


$T_1 = 1$ $T_2 = 3$ $T_3 = 6$ $T_4 = 10$

- Is this sequence arithmetic?
  No, since $3 - 1 = 2$ and $6 - 3 = 3 \neq 2$, so there is no common difference.
- Is the sequence geometric?

Motivation

Look at the sequence $(T_n)_{n\geq 1}$ which starts $1, 3, 6, 10, 15, \ldots$. These are called the triangular numbers since they represent the number of dots in an equilateral triangle (think of how you arrange 10 bowling pins: a row of 4 plus a row of 3 plus a row of 2 and a row of 1).

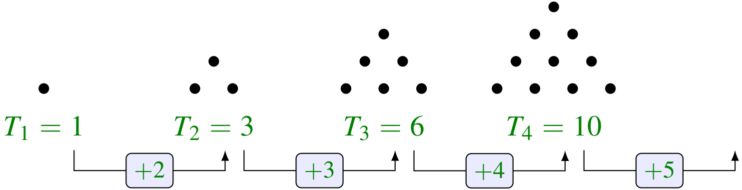


$T_1 = 1$ $T_2 = 3$ $T_3 = 6$ $T_4 = 10$

- Is this sequence arithmetic?
  No, since $3 - 1 = 2$ and $6 - 3 = 3 \neq 2$, so there is no common difference.
- Is the sequence geometric?
  No. $3/1 = 3$ but $6/3 = 2$, so there is no common ratio.

Motivation

Look at the sequence $(T_n)_{n\geq 1}$ which starts $1, 3, 6, 10, 15, \ldots$. These are called the triangular numbers since they represent the number of dots in an equilateral triangle (think of how you arrange 10 bowling pins: a row of 4 plus a row of 3 plus a row of 2 and a row of 1).



- Is this sequence arithmetic?
  No, since $3 - 1 = 2$ and $6 - 3 = 3 \neq 2$, so there is no common difference.
- Is the sequence geometric?
  No. $3/1 = 3$ but $6/3 = 2$, so there is no common ratio.
- Notice that the differences between terms generate an arithmetic sequence: $2, 3, 4, 5, 6, \ldots$.
  This says that the nth term of the triangular sequence is the sum of the first n terms in the sequence $1, 2, 3, 4, 5, \ldots$, i.e, the triangular sequence is a sequence of partial sums.

Summing Arithmetic Sequences: Reverse and Add I

Example 10

Find the sum: $2 + 5 + 8 + 11 + 14 + \cdots + 470$.

Solution. If we add the first and last terms, we get 472. The second term and second-to-last term also add up to 472. To keep track of everything, we might express this as follows. Call the sum S. Then,

$$\begin{array}{rcccccccccc} & S = & 2 & + & 5 & + & 8 & + \cdots + & 467 & + & 470 \\ + & S = & 470 & + & 467 & + & 464 & + \cdots + & 5 & + & 2 \\ \hline & 2S = & 472 & + & 472 & + & 472 & + \cdots + & 472 & + & 472 \end{array}$$

Hence, to find $2S$ then we add 472 to itself a number of times. What number?

We need to decide how many terms are in the sum. Since the terms form an arithmetic sequence, the nth term in the sum (counting 2 as the 0th term) can be expressed as $2 + 3n$. If $2 + 3n = 470$ then $n = 156$. So n ranges from 0 to 156, giving 157 terms in the sum. This is the number of 472's in the sum for $2S$. Thus

$$2S = 157 \times 472 = 74104 \quad \implies \quad S = \frac{74104}{2} = 37052$$

Summing Arithmetic Sequences: Reverse and Add I

Example 10

Find the sum: $2 + 5 + 8 + 11 + 14 + \cdots + 470$.

Solution. If we add the first and last terms, we get 472. The second term and second-to-last term also add up to 472. To keep track of everything, we might express this as follows. Call the sum S. Then,

$$\begin{array}{rcccccccccc} & S = & 2 & + & 5 & + & 8 & + \cdots + & 467 & + & 470 \\ + & S = & 470 & + & 467 & + & 464 & + \cdots + & 5 & + & 2 \\ \hline & 2S = & 472 & + & 472 & + & 472 & + \cdots + & 472 & + & 472 \end{array}$$

Hence, to find $2S$ then we add 472 to itself a number of times. What number?

We need to decide how many terms are in the sum. Since the terms form an arithmetic sequence, the nth term in the sum (counting 2 as the 0th term) can be expressed as $2 + 3n$. If $2 + 3n = 470$ then $n = 156$. So n ranges from 0 to 156, giving 157 terms in the sum. This is the number of 472's in the sum for $2S$. Thus

$$2S = 157 \times 472 = 74104 \quad \Longrightarrow \quad S = \frac{74104}{2} = 37052$$

Summing Arithmetic Sequences: Reverse and Add I

Example 10

Find the sum: $2 + 5 + 8 + 11 + 14 + \cdots + 470$.

Solution. If we add the first and last terms, we get 472. The second term and second-to-last term also add up to 472. To keep track of everything, we might express this as follows. Call the sum S. Then,

$$\begin{array}{rrcccccccccc} & S = & 2 & + & 5 & + & 8 & +\cdots+ & 467 & + & 470 \\ + & S = & 470 & + & 467 & + & 464 & +\cdots+ & 5 & + & 2 \\ \hline & 2S = & 472 & + & 472 & + & 472 & +\cdots+ & 472 & + & 472 \end{array}$$

Hence, to find $2S$ then we add 472 to itself a number of times. What number?

We need to decide how many terms are in the sum. Since the terms form an arithmetic sequence, the nth term in the sum (counting 2 as the 0th term) can be expressed as $2 + 3n$. If $2 + 3n = 470$ then $n = 156$. So n ranges from 0 to 156, giving 157 terms in the sum. This is the number of 472's in the sum for $2S$. Thus

$$2S = 157 \times 472 = 74104 \qquad \Longrightarrow \qquad S = \frac{74104}{2} = 37052$$

Summing Arithmetic Sequences: Reverse and Add I

Example 10

Find the sum: $2 + 5 + 8 + 11 + 14 + \cdots + 470$.

Solution. If we add the first and last terms, we get 472. The second term and second-to-last term also add up to 472. To keep track of everything, we might express this as follows. Call the sum S. Then,

$$\begin{array}{rrcrcrcrcrcr} & S = & 2 & + & 5 & + & 8 & +\cdots+ & 467 & + & 470 \\ + & S = & 470 & + & 467 & + & 464 & +\cdots+ & 5 & + & 2 \\ \hline & 2S = & 472 & + & 472 & + & 472 & +\cdots+ & 472 & + & 472 \end{array}$$

Hence, to find $2S$ then we add 472 to itself a number of times. What number?

We need to decide how many terms are in the sum. Since the terms form an arithmetic sequence, the nth term in the sum (counting 2 as the 0th term) can be expressed as $2 + 3n$. If $2 + 3n = 470$ then $n = 156$. So n ranges from 0 to 156, giving 157 terms in the sum. This is the number of 472's in the sum for $2S$. Thus

$$2S = 157 \times 472 = 74104 \implies S = \frac{74104}{2} = 37052$$

Summing Arithmetic Sequences: Reverse and Add II

The process covered in the previous slide will work for any sum of arithmetic sequences.

STEP 1 Call the sum S.

STEP 2 Reverse and add.

STEP 3 This produces a single number added to itself many times.

STEP 4 Determine the number of times.

STEP 5 Multiply. Divide by 2. Done

Definition 11 (Arithmetic Series)

The sum of the terms of the arithmetic sequence

$$S_n = [a] + [a+d] + [a+2d] + \cdots + [a+nd]$$

is called an arithmetic series and is given by

$$S_n = (n+1)a + \frac{dn(n+1)}{2}$$

Summing Arithmetic Sequences: Reverse and Add II

The process covered in the previous slide will work for any sum of arithmetic sequences.

- STEP 1 Call the sum S.
- STEP 2 Reverse and add.
- STEP 3 This produces a single number added to itself many times.
- STEP 4 Determine the number of times.
- STEP 5 Multiply. Divide by 2. Done

Definition 11 (Arithmetic Series)

The sum of the terms of the arithmetic sequence

$$S_n = [a] + [a+d] + [a+2d] + \cdots + [a+nd]$$

is called an arithmetic series and is given by

$$S_n = (n+1)a + \frac{dn(n+1)}{2}$$

Summing Geometric Sequences: Multiply and Subtract I

To find the sum of a geometric sequence, we cannot just reverse and add. Instead we multiply and subtract:

Example 12

What is $3 + 6 + 12 + 24 + \cdots + 12288$?

STEP 1 Call the sum S.

STEP 2 Multiply each term by the common ratio, $r = 2$

STEP 3 Subtract, and solve for S.

Summing Geometric Sequences: Multiply and Subtract I

To find the sum of a geometric sequence, we cannot just reverse and add. Instead we multiply and subtract:

Example 12

What is $3 + 6 + 12 + 24 + \cdots + 12288$?

This terms in the sum are from a geometric progression with initial term, $a_0 = 3$, and common ratio, $r = 2$.

STEP 1 Call the sum S.

STEP 2 Multiply each term by the common ratio, $r = 2$

STEP 3 Subtract, and solve for S.

Summing Geometric Sequences: Multiply and Subtract I

To find the sum of a geometric sequence, we cannot just reverse and add. Instead we multiply and subtract:

Example 12

What is $3 + 6 + 12 + 24 + \cdots + 12288$?

This terms in the sum are from a geometric progression with initial term, $a_0 = 3$, and common ratio, $r = 2$.

STEP 1 Call the sum S.

STEP 2 Multiply each term by the common ratio, $r = 2$

STEP 3 Subtract, and solve for S.

$$S = \quad 3 + \quad 6 + 12 + 24 + \cdots + 12288$$

Summing Geometric Sequences: Multiply and Subtract I

To find the sum of a geometric sequence, we cannot just reverse and add. Instead we multiply and subtract:

Example 12

What is $3 + 6 + 12 + 24 + \cdots + 12288$?

This terms in the sum are from a geometric progression with initial term, $a_0 = 3$, and common ratio, $r = 2$.

STEP 1 Call the sum S.

STEP 2 Multiply each term by the common ratio, $r = 2$

STEP 3 Subtract, and solve for S.

$$\begin{array}{rrrr} S = & 3 + & 6 + 12 + 24 + \cdots + 12288 & \\ 2S = & & 6 + 12 + 24 + \cdots + 12288 & +24576 \\ \hline \end{array}$$

Summing Geometric Sequences: Multiply and Subtract I

To find the sum of a geometric sequence, we cannot just reverse and add. Instead we multiply and subtract:

Example 12

What is $3 + 6 + 12 + 24 + \cdots + 12288$?

This terms in the sum are from a geometric progression with initial term, $a_0 = 3$, and common ratio, $r = 2$.

STEP 1 Call the sum S.

STEP 2 Multiply each term by the common ratio, $r = 2$

STEP 3 Subtract, and solve for S.

$$\begin{array}{rlll} S = & 3 + & 6 + 12 + 24 + \cdots + 12288 & \\ 2S = & & 6 + 12 + 24 + \cdots + 12288 & +24576 \\ \hline -S = & 3 + & 0 + 0 + 0 + \cdots + 0 & -24576 \end{array}$$

$$-S = 3 - 24576 \quad \Longrightarrow \quad S = 24573$$

Summing Geometric Sequences: Multiply and Subtract II

Definition 13 (Geometric Series)

The sum of the terms of the geometric sequence

$$S_n = [a] + [ar] + [ar^2] + \cdots + [ar^n]$$

is called a geometric series and is given by

$$S_n = \frac{a(1 - r^{n+1})}{1 - r}$$

- In the special case of $-1 < r < 1$ the terms in the geometric sequence tends towards zero fast enough that the sum of the series tends to the finite value

$$S_\infty = \lim_{n \to \infty} S_n = \lim_{n \to \infty} \frac{a}{1 - r}$$

since $r^{n+1} \to 0$ as $n \to \infty$.

Summing Geometric Sequences: Multiply and Subtract II

Definition 13 (Geometric Series)

The sum of the terms of the geometric sequence

$$S_n = [a] + [ar] + [ar^2] + \cdots + [ar^n]$$

is called a geometric series and is given by

$$S_n = \frac{a(1 - r^{n+1})}{1 - r}$$

- In the special case of $-1 < r < 1$ the terms in the geometric sequence tends towards zero fast enough that the sum of the series tends to the finite value

$$S_\infty = \lim_{n \to \infty} S_n = \lim_{n \to \infty} \frac{a}{1 - r}$$

since $r^{n+1} \to 0$ as $n \to \infty$.

Review Exercises 2 (Arithmetic and Geometric Progressions)

Question 1:

Consider the sequence $5, 9, 13, 17, 21, \ldots$ with $a_1 = 5$

- ⓐ Give a recursive definition for the sequence.
- ⓑ Give a closed formula for the nth term of the sequence.
- ⓒ Is 2013 a term in the sequence? Explain.
- ⓓ How many terms does the sequence $5, 9, 13, 17, 21, \ldots, 533$ have?
- ⓔ Determine the sum: $5 + 9 + 13 + 17 + 21 + \cdots + 533$. Show your work.
- ⓕ Use what you found above to find b_n, the n^{th} term of $1, 6, 15, 28, 45, \ldots$, where $b_0 = 1$

Outline

Math vs. Programming (Python/Processing/Java/. . .)

Computers are finite

In mathematics we can define a sequence, just like

$$a_n = 2^n, \text{for } n \geq 0 \qquad 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \ldots$$

and have no concerns that it has infinite length or that the values become arbitrary large. This is not the case when programming — we (or the language designers) need to deal with both of these issues.

Infinite length sequences in Mathematics → (usually) Finite length sequences in Python

Programmers need standard tasks/operations

- **Indexing** — Each position in a sequence is given a unique position/index, so we can access/change a single element by referring to its index.
- **Slicing** — Given a sequence collection we want to create a copy of part of that sequence.
- **Iterating over** — Looping over all elements (`for` loops and **list comprehensions**).
- **Filtering** — Given a collection construct a new collection containing only elements that satisfy a condition.

Python Implementation — Sets vs Lists

```python
S = set()   # cannot use {}
L = []      # here we can use list() or []
print(S, L)

S.add(3)        # we ADD to a set
L.append(3)     # but we APPEND to END of list
print(S, L)

S.add(3)    # elements are distinct
L.append(3)
print(S, L)

S.add("Hello")    # can store mixture of data types
L.append("Hello") # can store mixture of data types
print(S,L)

S.add("All")    # unordered
L.append("All") # ordered
print(S,L)
```

Python Implementation — Sets vs Lists

```python
S = set()    # cannot use {}
L = []       # here we can use list() or []
print(S, L)

S.add(3)        # we ADD to a set
L.append(3)     # but we APPEND to END of list
print(S, L)

S.add(3)     # elements are distinct
L.append(3)
print(S, L)

S.add("Hello")     # can store mixture of data types
L.append("Hello") # can store mixture of data types
print(S,L)

S.add("All")     # unordered
L.append("All") # ordered
print(S,L)
```

```
set() []
```

Python Implementation — Sets vs Lists

```python
S = set()    # cannot use {}
L = []       # here we can use list() or []
print(S, L)

S.add(3)        # we ADD to a set
L.append(3)     # but we APPEND to END of list
print(S, L)

S.add(3)    # elements are distinct
L.append(3)
print(S, L)

S.add("Hello")    # can store mixture of data types
L.append("Hello") # can store mixture of data types
print(S,L)

S.add("All")    # unordered
L.append("All") # ordered
print(S,L)
```

```
set() []
{3} [3]
```

Python Implementation — Sets vs Lists

```
S = set()   # cannot use {}
L = []      # here we can use list() or []
print(S, L)

S.add(3)       # we ADD to a set
L.append(3)    # but we APPEND to END of list
print(S, L)

S.add(3)    # elements are distinct
L.append(3)
print(S, L)

S.add("Hello")    # can store mixture of data types
L.append("Hello") # can store mixture of data
print(S,L)

S.add("All")    # unordered
L.append("All") # ordered
print(S,L)
```

```
set() []
{3} [3]
{3} [3, 3]
```

Python Implementation — Sets vs Lists

```python
S = set()    # cannot use {}
L = []       # here we can use list() or []
print(S, L)

S.add(3)        # we ADD to a set
L.append(3)     # but we APPEND to END of list
print(S, L)

S.add(3)     # elements are distinct
L.append(3)
print(S, L)

S.add("Hello")     # can store mixture of data
L.append("Hello") # can store mixture of data
print(S,L)

S.add("All")     # unordered
L.append("All") # ordered
print(S,L)
```

```
set() []
{3} [3]
{3} [3, 3]
{'Hello', 3} [3, 3, 'Hello']
```

Python Implementation — Sets vs Lists

```
S = set()   # cannot use {}
L = []      # here we can use list() or []
print(S, L)

S.add(3)       # we ADD to a set
L.append(3)    # but we APPEND to END of list
print(S, L)

S.add(3)    # elements are distinct
L.append(3)
print(S, L)

S.add("Hello")    # can store mixture of data
L.append("Hello") # can store mixture of data
print(S,L)

S.add("All")     # unordered
L.append("All") # ordered
print(S,L)
```

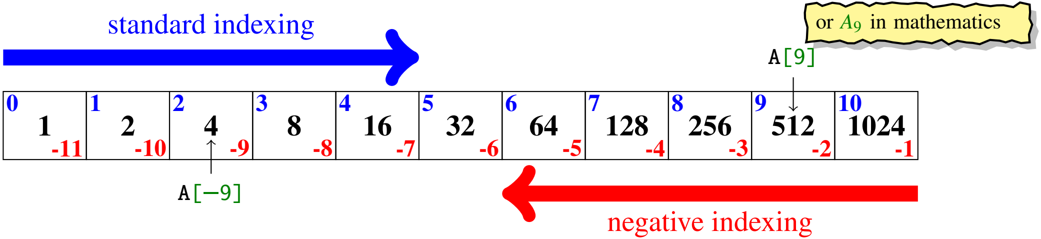
```
set() []
{3} [3]
{3} [3, 3]
{'Hello', 3} [3, 3, 'Hello']
{'Hello', 'All', 3} [3, 3, 'Hello', 'All']
```

Indexing

To help illustrate indexing we will define a list containing the powers of 2 up to and including 2^{10}.

```
A = [1,2,4,8,16,32,64,128,256,512,1024]
```

- The collection (a `list`) is **ordered** — so we can talk about which data value (item/element) comes before/after another data value.
- In addition, each data value has a position, called **index**, which counts from the left of the list. Python is zero-based language so index starts at zero.
- Python, also support **negative indexing** which counts backwards from the end of the list.



standard indexing

or A_9 in mathematics

`A[9]`

0	1	2	3	4	5	6	7	8	9	10
1	2	4	8	16	32	64	128	256	512	1024
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`A[−9]`

negative indexing

Slicing

Definition 14 (Slicing)

Slicing is a compact syntax to construct a sub-sequence collection from a larger collection.
A slice consists of

`[start:end:step]`

where

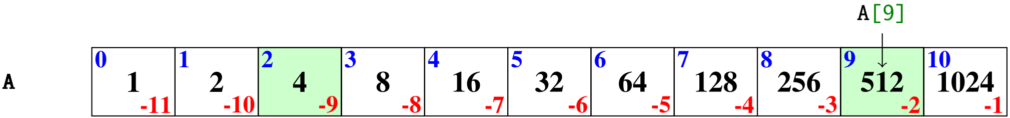
- `start` — the starting index (inclusive). Defaults to 0 (i.e., start of the collection) if omitted.
- `end` — the ending index (exclusive). Defaults to length of collection if omitted.
- `step` — the amount by which the index increases, defaults to 1. If it's negative, you're slicing over the collection in reverse.

Some common slices:

Given collection, `A`, then

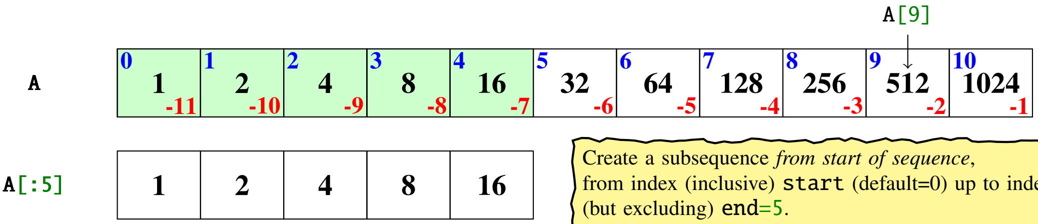
- `A[:]` creates a copy of the entire collection. (uses default value for start, end, and step)
- `A[::-1]` creates a copy of the entire collection in reverse (step=-1 reverses the collection)

Slicing II



0	1	2	3	4	5	6	7	8	9	10
1	2	4	8	16	32	64	128	256	512	1024
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

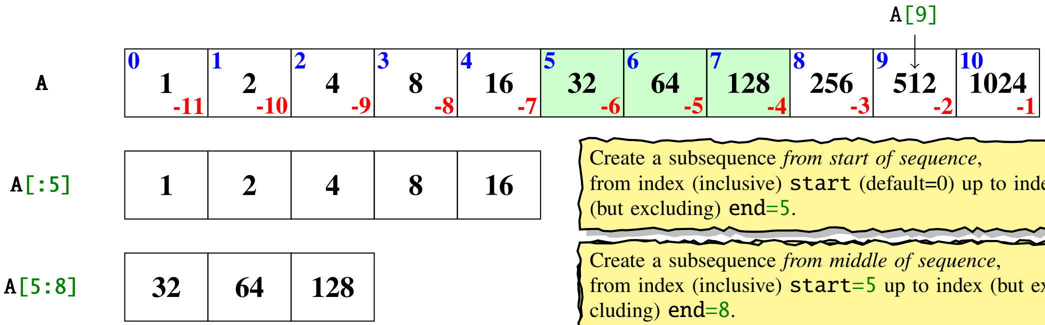
Slicing II



	0	1	2	3	4	5	6	7	8	9	10
`A`	1	2	4	8	16	32	64	128	256	512	1024
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
`A[:5]`	1	2	4	8	16						

Create a subsequence *from start of sequence*, from index (inclusive) `start` (default=0) up to index (but excluding) `end=5`.

Slicing II

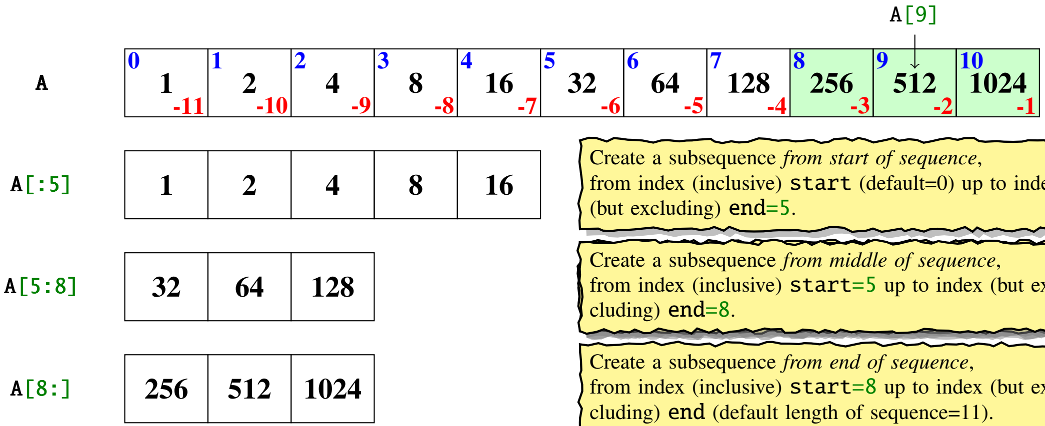


	0	1	2	3	4	5	6	7	8	9	10
A	1	2	4	8	16	32	64	128	256	512	1024
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
`A[:5]`	1	2	4	8	16						
`A[5:8]`	32	64	128								

Create a subsequence *from start of sequence*, from index (inclusive) `start` (default=0) up to index (but excluding) `end=5`.

Create a subsequence *from middle of sequence*, from index (inclusive) `start=5` up to index (but excluding) `end=8`.

Slicing II



	0	1	2	3	4	5	6	7	8	9	10
`A`	1	2	4	8	16	32	64	128	256	512	1024
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`A[:5]`	1	2	4	8	16

`A[5:8]`	32	64	128

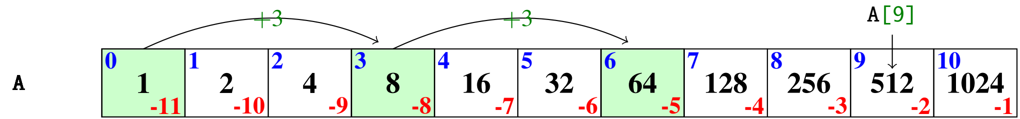
`A[8:]`	256	512	1024

Create a subsequence *from start of sequence*, from index (inclusive) `start` (default=0) up to index (but excluding) `end=5`.

Create a subsequence *from middle of sequence*, from index (inclusive) `start=5` up to index (but excluding) `end=8`.

Create a subsequence *from end of sequence*, from index (inclusive) `start=8` up to index (but excluding) `end` (default length of sequence=11).

Slicing II



	0	1	2	3	4	5	6	7	8	9	10
`A`	1	2	4	8	16	32	64	128	256	512	1024
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`A[:5]`	1	2	4	8	16

Create a subsequence *from start of sequence*, from index (inclusive) `start` (default=0) up to index (but excluding) `end`=5.

`A[5:8]`	32	64	128

Create a subsequence *from middle of sequence*, from index (inclusive) `start`=5 up to index (but excluding) `end`=8.

`A[8:]`	256	512	1024

Create a subsequence *from end of sequence*, from index (inclusive) `start`=8 up to index (but excluding) `end` (default length of sequence=11).

`A[0:9:3]`	1	8	64

Create a subsequence from index (inclusive) `start`=0 up to index (but excluding) `end`=9 using a `step`=3.

Iterating over Collections

- Python’s `for` is used to iterate over elements in a collections.
- Function `enumerate` counts the elements during iteratation.

```python
A = [1,2,4,8,16,32,64,128,256,512,1024]

# loop over all elements
for value in A:
    print(value)

# count and looping over all elements
for pos, value in enumerate(A):
    print(pos, value)

# loop over all positions - rarely used in python
for pos in range(len(A)):
    print(pos, A[pos])
```

Iterating over Collections

- Python’s **`for`** is used to iterate over elements in a collections.
- Function **`enumerate`** counts the elements during iteratation.

```
A = [1,2,4,8,16,32,64,128,256,512,1024]

# loop over all elements
for value in A:
    print(value)

# count and looping over all elements
for pos, value in enumerate(A):
    print(pos, value)

# loop over all positions - rarely used in pyt
for pos in range(len(A)):
    print(pos, A[pos])
```

```
1
2
4
8
16
32
64
128
256
512
1024
```

Iterating over Collections

- Python’s **for** is used to iterate over elements in a collections.
- Function **enumerate** counts the elements during iteratation.

```python
A = [1,2,4,8,16,32,64,128,256,512,1024]

# loop over all elements
for value in A:
    print(value)

# count and looping over all elements
for pos, value in enumerate(A):
    print(pos, value)

# loop over all positions - rarely used in pyt
for pos in range(len(A)):
    print(pos, A[pos])
```

```
1
2
4
8
16
32
64
128
256
512
1024
```

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
10 1024
```

Filtering

Definition 15 (Filtering)

Build a collection from another by selecting (**filtering**) elements in the collection that satisfy some criteria.

Task — Given a list of powers of 2, select all values that have remainder 4 when divided by 10:

```python
A = [1,2,4,8,16,32,64,128,256,512,1024]

# old style filtering
B = []
for value in A:
    if value % 10==4:   # remainder is 4
        B.append(value)
print(B)

# or using list comprehension
B = [value for value in A if value % 10==4]
print(B)
```

Filtering

Definition 15 (Filtering)

Build a collection from another by selecting (**filtering**) elements in the collection that satisfy some criteria.

Task — Given a list of powers of 2, select all values that have remainder 4 when divided by 10:

```python
A = [1,2,4,8,16,32,64,128,256,512,1024]

# old style filtering
B = []
for value in A:
    if value % 10==4:   # remainder is 4
        B.append(value)
print(B)

# or using list comprehension
B = [value for value in A if value % 10==4]
print(B)
```

```
[4, 64, 1024]
[4, 64, 1024]
```

Filtering

Definition 15 (Filtering)

Build a collection from another by selecting (**filtering**) elements in the collection that satisfy some criteria.

Task — Given a list of powers of 2, select all values that have remainder 4 when divided by 10:

```python
A = [1,2,4,8,16,32,64,128,256,512,1024]

# old style filtering
B = []
for value in A:
    if value % 10==4:  # remainder is 4
        B.append(value)
print(B)

# or using list comprehension
B = [value for value in A if value % 10==4]
print(B)
```

Create empty list. Loop over original.
If element satisfies criteria, then append it to list.

```
[4, 64, 1024]
[4, 64, 1024]
```

Filtering

Definition 15 (Filtering)

Build a collection from another by selecting (**filtering**) elements in the collection that satisfy some criteria.

Task — Given a list of powers of 2, select all values that have remainder 4 when divided by 10:

```python
A = [1,2,4,8,16,32,64,128,256,512,1024]

# old style filtering
B = []
for value in A:
    if value % 10==4:  # remainder is 4
        B.append(value)
print(B)

# or using list comprehension
B = [value for value in A if value % 10==4]
print(B)
```

Create empty list. Loop over original.
If element satisfies criteria, then append it to list.

List comprehension

```
[4, 64, 1024]
[4, 64, 1024]
```

Filtering

Definition 15 (Filtering)

Build a collection from another by selecting (**filtering**) elements in the collection that satisfy some criteria.

Task — Given a list of powers of 2, select all values that have remainder 4 when divided by 10:

```
A = [1,2,4,8,16,32,64,128,256,512,1024]

# old style filtering
B = []
for value in A:
    if value % 10==4:   # remainder is 4
        B.append(value)
print(B)

# or using list comprehension
B = [value for value in A if value % 10==4]
print(B)
```

Create empty list. Loop over original.
If element satisfies criteria, then append it to list.

List comprehension

```
[4, 64, 1024]
[4, 64, 1024]
```

List comprehension

Definition 16 (List comprehension)

List comprehension is a compact syntax to construct a new sequence from another collection
It consists of

```
[EXPRESSION for value in COLLECTION if CONDITION]
```

where

- `EXPRESSION` is any python expression.
- `COLLECTION` is any python collection (set, list, . . .)
- `CONDITION` — is python expression that results in `True` or `False`

- As a programmer you don't have to use list comprehensions and instead use the longer traditional style, but you will need to be able to read and understand it since it is the default style in modern Python programmers.
- Replacing `[` and `]` by `{` and `}` will create a `set` instead of a new `list`.

List Comprehension Example 1

Task — Create list of first 10 square numbers from the set of natural numbers (ℕ).

26

```python
# traditional approach
squares = []
for k in range(10):
    squares.append(k**2)
print(squares)

# using list comprehension
squares = [k**2 for k in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension Example 1

Task — Create list of first 10 square numbers from the set of natural numbers ($\mathbb{N}$).

```python
# traditional approach
squares = []
for k in range(10):
    squares.append(k**2)
print(squares)

# using list comprehension
squares = [k**2 for k in range(10)]
print(squares)
```

Create empty list. Loop over original collection, calculate expression (`k**2`) and append result to list.

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension Example 1

Task — Create list of first 10 square numbers from the set of natural numbers (ℕ).

```python
# traditional approach
squares = []
for k in range(10):
    squares.append(k**2)
print(squares)

# using list comprehension
squares = [k**2 for k in range(10)]
print(squares)
```

Create empty list. Loop over original collection, calculate expression (`k**2`) and append result to list.

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension Example 1

Task — Create list of first 10 square numbers from the set of natural numbers (ℕ).

```python
# traditional approach
squares = []
for k in range(10):
    squares.append(k**2)
print(squares)

# using list comprehension
squares = [k**2 for k in range(10)]
print(squares)
```

Create empty list. Loop over original collection, calculate expression (`k**2`) and append result to list.

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- The COLLECTION is `range(10)` which generates the list `[0,1,2,3,4,5,6,7,8,9]`.
- The EXPRESSION is `k**2` which generates the required pattern.
- There is no CONDITION so all elements in COLLECTION are used.

List Comprehension Example 2

Task — Create list of even integers up to but not including 10.

30

```python
# traditional approach
evens = []
for k in range(10):
    if k % 2 == 0:
        evens.append(k)
print(evens)

# using list comprehension
evens = [k for k in range(10) if k % 2 == 0]
print(evens)
```

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

List Comprehension Example 2

Task — Create list of even integers up to but not including 10.

```
# traditional approach
evens = []
for k in range(10):
    if k % 2 == 0:
        evens.append(k)
print(evens)

# using list comprehension
evens = [k for k in range(10) if k % 2 == 0]
print(evens)
```

Create empty list. Loop over original collection, if value matches criteria (even), then append value to list.

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

List Comprehension Example 2

Task — Create list of even integers up to but not including 10.

```python
# traditional approach
evens = []
for k in range(10):
    if k % 2 == 0:
        evens.append(k)
print(evens)

# using list comprehension
evens = [k for k in range(10) if k % 2 == 0]
print(evens)
```

Create empty list. Loop over original collection, if value matches criteria (even), then append value to list.

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

List Comprehension Example 2

Task — Create list of even integers up to but not including 10.

```
# traditional approach
evens = []
for k in range(10):
    if k % 2 == 0:
        evens.append(k)
print(evens)

# using list comprehension
evens = [k for k in range(10) if k % 2 == 0]
print(evens)
```

Create empty list. Loop over original collection, if value matches criteria (even), then append value to list.

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

- The COLLECTION is `range(10)` which generates the list `[0,1,2,3,4,5,6,7,8,9]`.
- The EXPRESSION is `k` which generates the required pattern.
- The CONDITION, `k%2==0` selects the even integers only.

List Comprehension Example 3

Task — Create list of the length of each word in a list of words.

34

```python
names = ['Alice', 'Bob', 'Charlie']

# traditional approach
lengths = []
for name in names:
    lengths.append(len(name))
print(lengths)

# using list comprehension
lengths = [len(name) for name in names]
print(lengths)
```

```
[5, 3, 7]
[5, 3, 7]
```

List Comprehension Example 3

Task — Create list of the length of each word in a list of words.

```python
names = ['Alice', 'Bob', 'Charlie']

# traditional approach
lengths = []
for name in names:
    lengths.append(len(name))
print(lengths)

# using list comprehension
lengths = [len(name) for name in names]
print(lengths)
```

Create empty list. Loop over original collection, calculate length of word, then append result to list.

```
[5, 3, 7]
[5, 3, 7]
```

List Comprehension Example 3

Task — Create list of the length of each word in a list of words.

```python
names = ['Alice', 'Bob', 'Charlie']

# traditional approach
lengths = []
for name in names:
    lengths.append(len(name))
print(lengths)

# using list comprehension
lengths = [len(name) for name in names]
print(lengths)
```

Create empty list. Loop over original collection, calculate length of word, then append result to list.

```
[5, 3, 7]
[5, 3, 7]
```

List Comprehension Example 3

Task — Create list of the length of each word in a list of words.

```python
names = ['Alice', 'Bob', 'Charlie']

# traditional approach
lengths = []
for name in names:
    lengths.append(len(name))
print(lengths)

# using list comprehension
lengths = [len(name) for name in names]
print(lengths)
```

Create empty list. Loop over original collection, calculate length of word, then append result to list.

```
[5, 3, 7]
[5, 3, 7]
```

- The COLLECTION is `names`, a list of strings.
- The EXPRESSION is `len(name)` which computes the length of the string stored in `name`.
- There is no CONDITION so all elements in COLLECTION are used.

Aside - Tuples

Definition 17 (Tuple)

A **tuple** is ordered, immutable collection.

- A **immutable** collection is unchangeable, meaning that we cannot change, add or remove items after the collection has been created.
- Tuple are denoted by round brackets, `(` and `)`.
- Unfortunately, round brackets are also used in controlling the order of operations in expressions. So a tuple with just one element requires a comma.

38

```python
fruits = ("apple", "banana", "cherry")
print(fruits)

fruits = ("apple",)
print(fruits)
```

```
('apple', 'banana', 'cherry')
('apple',)
```

Outline

Strings

Definition 18 (string `str`)

A **str** is a sequence collection consisting of a sequence of characters, like letters, numbers, and symbols.

Since a `str` is a sequence collection, all of the sequence operations we covered in `lists` also apply to `str`

Slight change in notes — we will come back to this section after functions.