# Pythonic Syntax

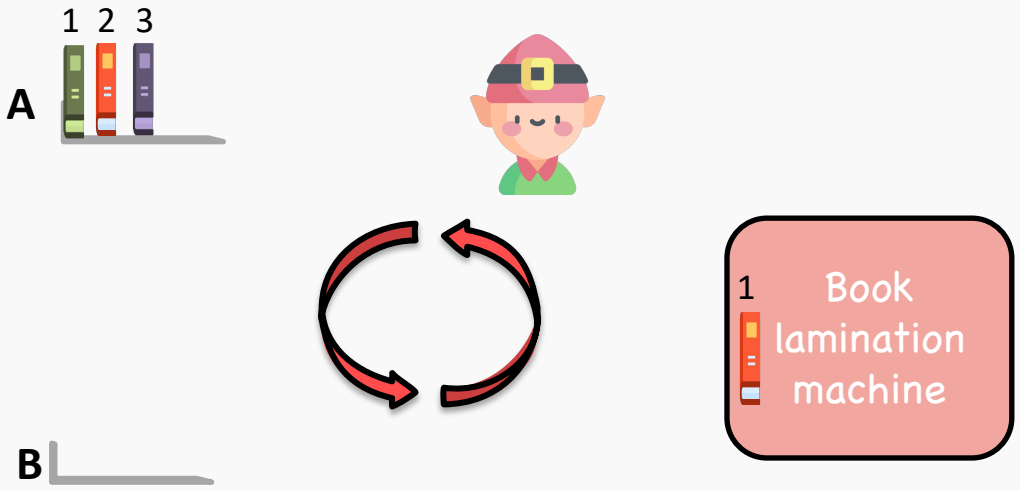
# List Comprehension

- List comprehensions simplify creation of lists by allowing the condition to be a part of list creation.
- This skips using loops and appending items to an existing list.
- List comprehensions also allow for conditionals to be a part of the defining expression.

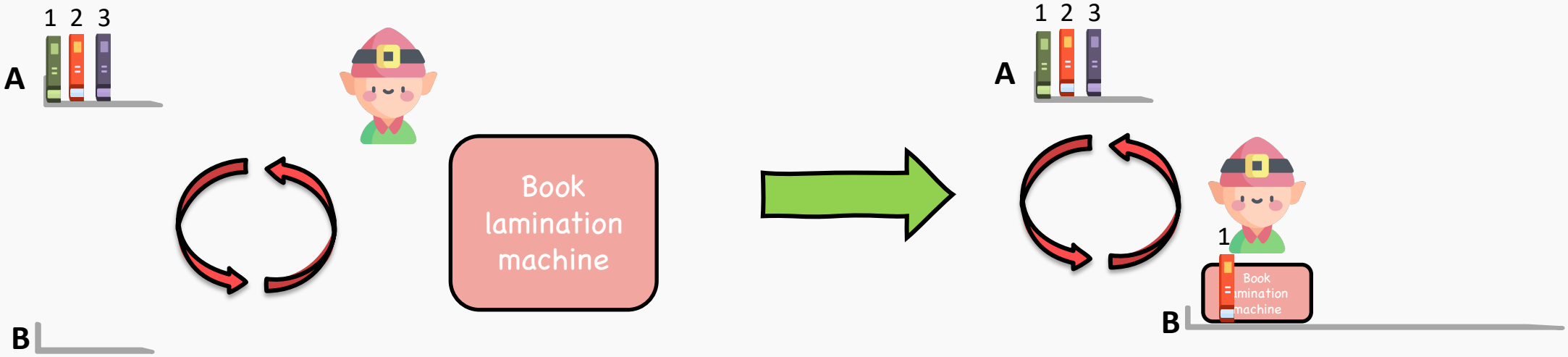
```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get
 a list containing their # squares
>>> B = [ number**2 for number in A ]

>>> print(B)
[2,4,9,16]
```

# List Comprehension

**List Comprehensions: A Pythonic way for making lists and loops.**



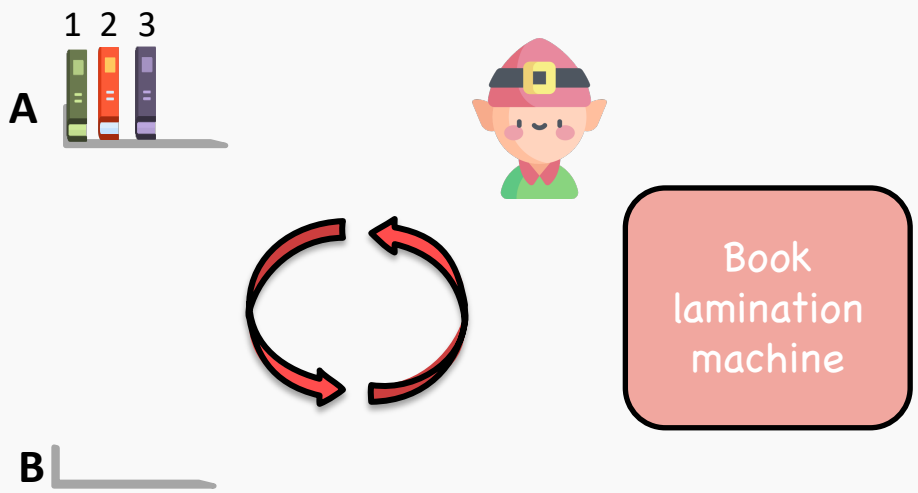
# List Comprehension

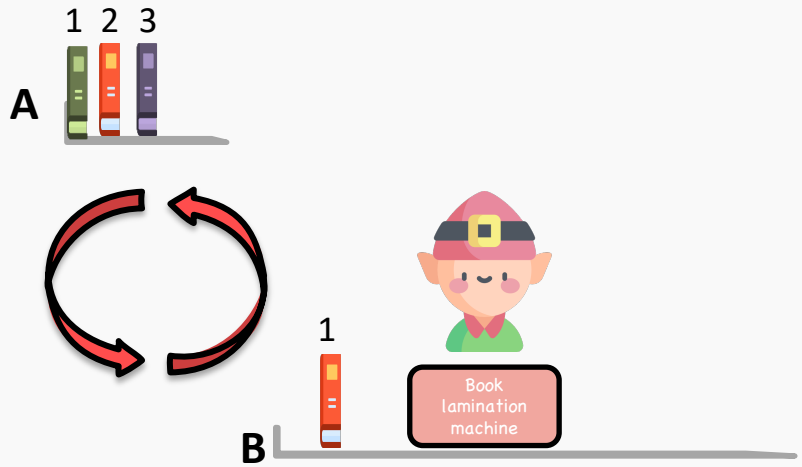


# List Comprehension

```
In [2]: laminated_list = []

In [3]: for book in range(books):
   ...:     laminated_book = f_lamination(book)
   ...:     laminated_list.append(book)
```

```
In [5]: laminated_list =
[f_lamination(book) for book in books]
```



# Dictionary Comprehension

**List Comprehensions: A Pythonic way for making lists and loops.**

```
List = [expression for item in iterable]
```

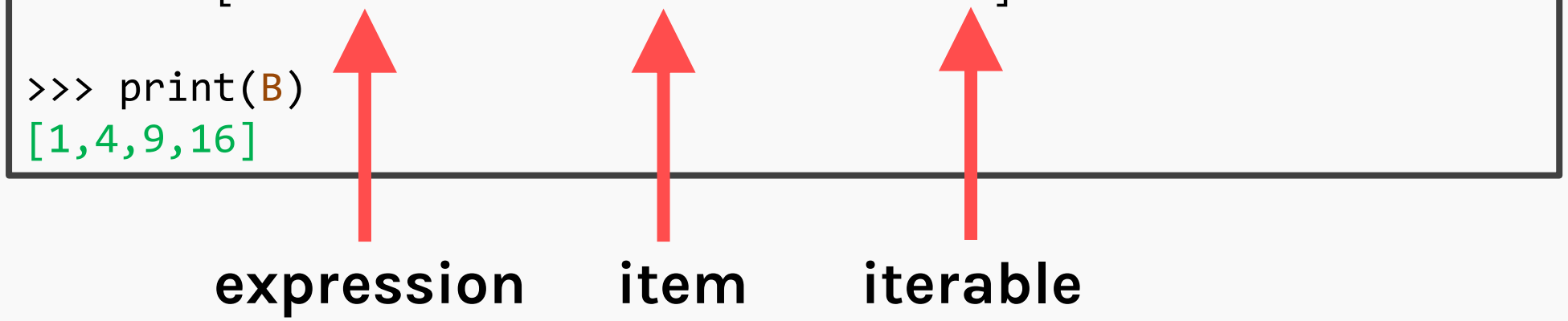
```
List = [expression for item in iterable if conditional]
```

```
List = [expression1 (if conditional) else expression2 for
                    item in iterable]
```

# List Comprehension

```
List = [expression for item in iterable]
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their #
squares
>>> B = [ number**2   for number in    A ]

>>> print(B)
[1,4,9,16]
```

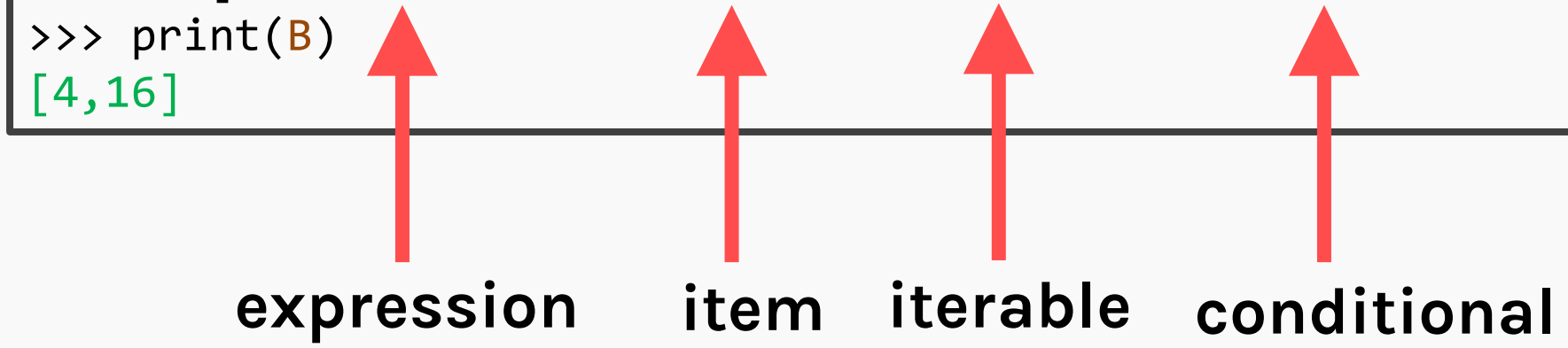


# List Comprehension

**List Comprehension with if conditional**

```
List = [expression1 for item in iterable  if conditional]
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their #
squares
>>>B = [ number**2 for  number    in A     if number %2 == 0]
>>> print(B)
[4,16]
```



# List Comprehension

**List Comprehension with if/else conditional**

```
List = [expression1 (if conditional) else expression2 for item in iterable]
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their #
squares
>>>B = [ number**2    if number%2 == 0 else 2*number for number in A ]
>>> print(B)
[2,4,6,16]
```

# Dictionary Comprehension

## Dictionary comprehension

Dict Comprehension works the same way as  list comprehension.
Only difference is using {} and includes a key:value.

```
Dict = {key:expression for item in iterable}
```

```
Dict = {key:expression for item in iterable if conditional}
```

```
Dict = {key : expression1 (if conditional) else expression2
                for item in iterable}
```

# Dictionary Comprehension

```
Dict = {key:expression for item in iterable}
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their #
squares
>>> B = {number:number**2   for number in   A ]

>>> print(B)
{1:1,2:4,3:9,4:16}
```

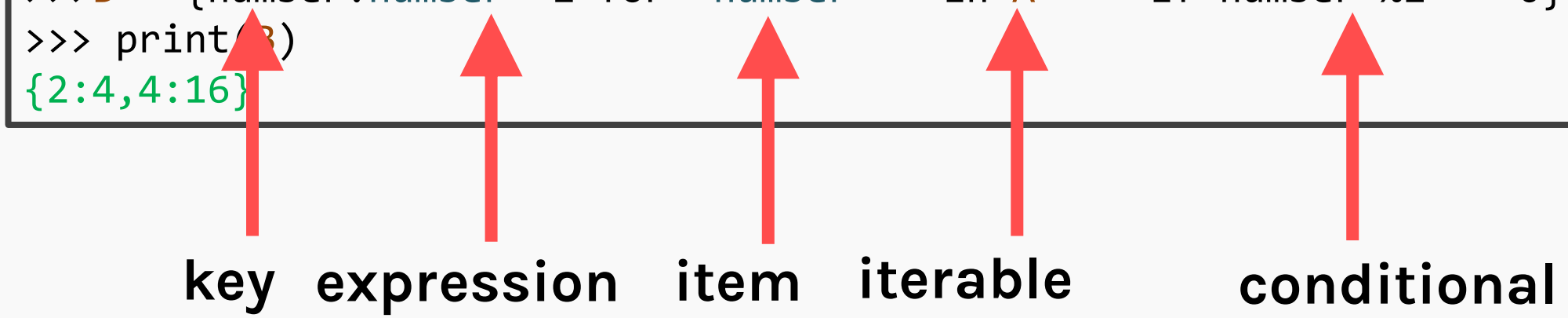
**key** **expression** **item** **iterable**

# Dictionary Comprehension

**Dictionary comprehension with if conditional**

```
Dict = {key:expression for item in iterable if conditional}
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their #
squares
>>>B = {number:number**2 for  number    in A     if number %2 == 0}
>>> print(B)
{2:4,4:16}
```

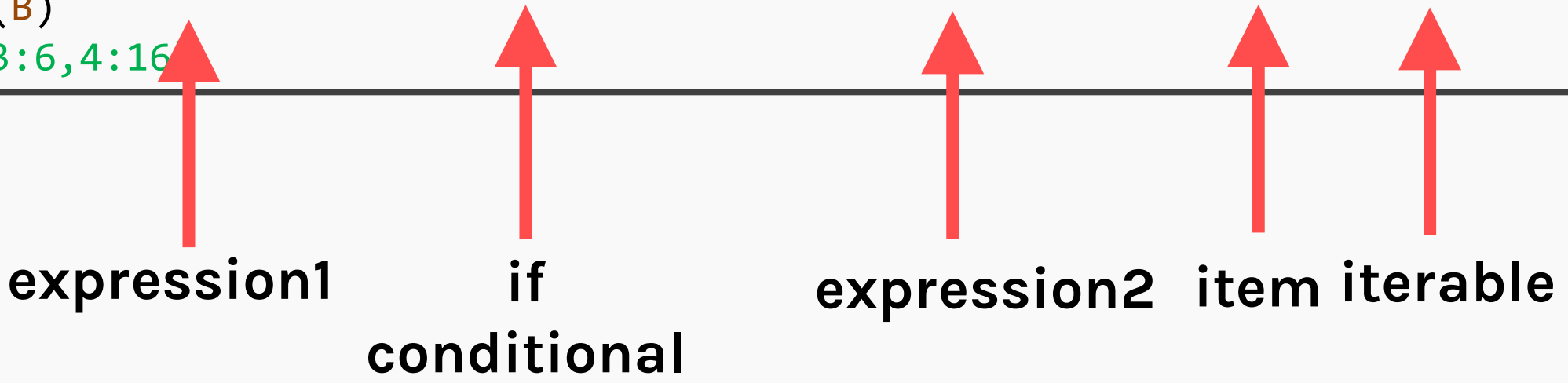


# Dictionary Comprehension

**Dictionary comprehension with if/else conditional**

```
Dict = {key :expression1 (if conditional) else expression2
                    for item in iterable}
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their # squares
>>>B ={number:(number**2 if number%2 == 0 else 2*number) for number in A }
>>> print(B)
{1:2,2:4,3:6,4:16}
```



# When not to use Comprehension

- A list comprehension in Python works by loading the entire output list into memory
- When the … problematic … a generator expression … list comprehension in Py…
- A **generator expression** doesn't create a single, large data structure in memory, but instead returns an iterable

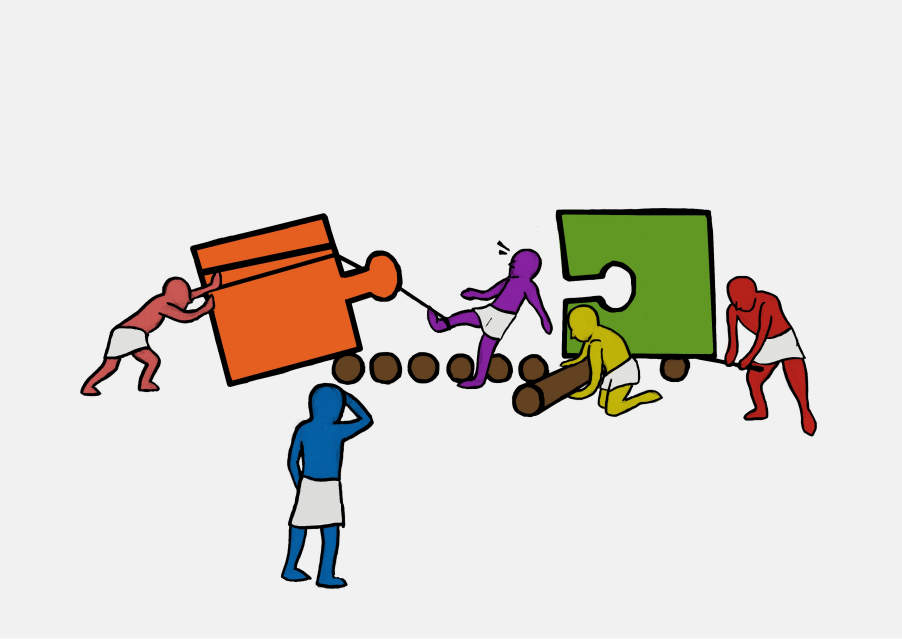
What is a generator expression ?

When do I use a generator instead of a list ?

```
# Using list comprehensions
>>> sum([i * i for i in
range(100000000000)])
Memory error

…g generator expressions
>>> sum(i * i for i in
range(100000000000))
333333333283333333350000
```

SYNTAX: (expression for item in iterable)

# Exercise



# 🏋️ Exercise: The Book Keeper's Apprentice - List & Dict Comprehensions

This exercise aims to get comfortable using list and dictionary comprehension for carrying out looping operations.

In this exercise, you are tasked to perform a set of operations involving list and dictionary comprehension on the dictionary you created in the previous exercise.

We will load data for this exercise that is stored in pickle format. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream. Read more about pickle files here.