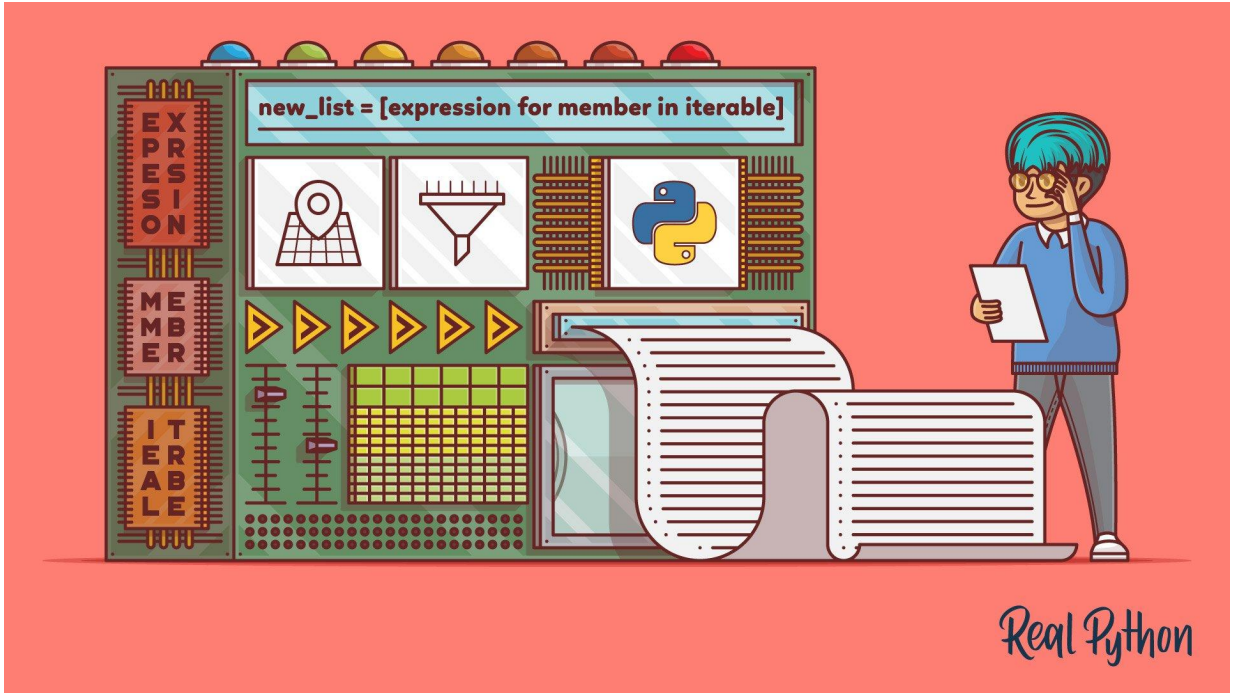
UNDERSTANDING PYTHON LIST COMPREHENSIONS



WHAT YOU WILL LEARN

1. **Rewrite loops and map() calls as a list comprehension in Python**
2. Choose between comprehensions, loops, and map() calls
3. Supercharge your comprehensions with conditional logic
4. Use comprehensions to replace filter()
5. Profile your code to solve performance questions

WHAT YOU WILL LEARN

1. Rewrite loops and map() calls as a list comprehension in Python
2. **Choose between comprehensions, loops, and map() calls**
3. Supercharge your comprehensions with conditional logic
4. Use comprehensions to replace filter()
5. Profile your code to solve performance questions

WHAT YOU WILL LEARN

1. Rewrite loops and map() calls as a list comprehension in Python
2. Choose between comprehensions, loops, and map() calls
3. **Supercharge your comprehensions with conditional logic**
4. Use comprehensions to replace filter()
5. Profile your code to solve performance questions

WHAT YOU WILL LEARN

1. Rewrite loops and map() calls as a list comprehension in Python
2. Choose between comprehensions, loops, and map() calls
3. Supercharge your comprehensions with conditional logic
4. **Use comprehensions to replace filter()**
5. Profile your code to solve performance questions

WHAT YOU WILL LEARN

1. Rewrite loops and map() calls as a list comprehension in Python
2. Choose between comprehensions, loops, and map() calls
3. Supercharge your comprehensions with conditional logic
4. Use comprehensions to replace filter()
5. **Profile your code to solve performance questions**

HOW TO CREATE LISTS IN PYTHON

- **Using for Loops**

HOW TO CREATE LISTS IN PYTHON

- **Using for Loops**
  1. Instantiate an empty list

HOW TO CREATE LISTS IN PYTHON

- **Using for Loops**
  1. Instantiate an empty list
  2. Loop over an iterable or range of elements

HOW TO CREATE LISTS IN PYTHON

- **Using for Loops**
  1. Instantiate an empty list
  2. Loop over an iterable or range of elements
  3. Append each element to the end of the list

HOW TO CREATE LISTS IN PYTHON

- **Using for Loops**
  1. Instantiate an empty list
  2. Loop over an iterable or range of element
  3. Append each element to the end of the list
- **Using map() Objects**
  - pass in a function and an iterable, and map() will create an object containing the output

HOW TO CREATE LISTS IN PYTHON

- **Using List Comprehensions**
  - simply define the list and its contents at the same time

HOW TO CREATE LISTS IN PYTHON

- **Using List Comprehensions**
  - simply define the list and its contents at the same time
  - new_list = [expression for member in iterable]

BENEFITS OF USING LIST COMPREHENSIONS

- Often described as more Pythonic

BENEFITS OF USING LIST COMPREHENSIONS

- Often described as more Pythonic
- A single tool that you can use in many different situations

BENEFITS OF USING LIST COMPREHENSIONS

- Often described as more Pythonic
- A single tool that you can use in many different situations
- Don’t need to remember the proper order of arguments like you would when you call map()

BENEFITS OF USING LIST COMPREHENSIONS

- Often described as more Pythonic
- A single tool that you can use in many different situations
- Don’t need to remember the proper order of arguments like you would when you call map()
- More declarative than loops, which means they’re easier to read and understand

USING CONDITIONAL LOGIC

```
New_list_1 = [expression for member in iterable]
```

USING CONDITIONAL LOGIC

```
New_list_1 = [expression for member in iterable]


New_list_2 = [expression for member in iterable (if conditional)]
```

The most common way to add conditional logic to a list comprehension is to add a conditional to the end of the expression

USING CONDITIONAL LOGIC

```
New_list_1 = [expression for member in iterable]

New_list_2 = [expression for member in iterable (if conditional)]

New_list_3 = [expression (if conditional) for member in iterable]
```

Use conditional logic to select from multiple possible output options

USING SET AND DICTIONARY COMPREHENSIONS

- A set comprehension is almost exactly the same as a list comprehension in Python

USING SET AND DICTIONARY COMPREHENSIONS

- A set comprehension is almost exactly the same as a list comprehension in Python
- Set comprehensions make sure the output contains no duplicates

USING SET AND DICTIONARY COMPREHENSIONS

- A set comprehension is almost exactly the same as a list comprehension in Python
- Set comprehensions make sure the output contains no duplicates
- You can create a set comprehension by using curly braces { } instead of brackets [ ]

USING THE WALRUS OPERATOR

- Python 3.8 introduced the assignment expression, also known as the ‘walrus operator’

USING THE WALRUS OPERATOR

- Python 3.8 introduced the assignment expression, also known as the ‘walrus operator’
- It allows you to run an expression while simultaneously assigning the output value to a variable.

WHEN NOT TO USE LIST COMPREHENSIONS

- Watch Out for Nested Comprehensions

WHEN NOT TO USE LIST COMPREHENSIONS

- Watch Out for Nested Comprehensions
- Choose Generators for Large Datasets

WHEN NOT TO USE LIST COMPREHENSIONS

- Watch Out for Nested Comprehensions
- Choose Generators for Large Datasets
- Profile to Optimize Performance

WHAT YOU HAVE LEARNED

- Simplify loops and map() calls with declarative list comprehensions

WHAT YOU HAVE LEARNED

- Simplify loops and map() calls with declarative list comprehensions
- Supercharge your comprehensions with conditional logic

WHAT YOU HAVE LEARNED

- Simplify loops and map() calls with declarative list comprehensions
- Supercharge your comprehensions with conditional logic
- Create set and dictionary comprehensions

WHAT YOU HAVE LEARNED

- Simplify loops and map() calls with declarative list comprehensions
- Supercharge your comprehensions with conditional logic
- Create set and dictionary comprehensions
- Determine when code clarity or performance dictates an alternative approach