List comprehensions (Sk■adanie listy)

Now let's talk about List comprehensions (Sk■adanie listy). List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Core syntax / Podstawowa sk■adnia

Now let's talk about Core syntax / Podstawowa sk■adnia. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Examples / Przyk■ady

Now let's talk about Examples / Przyk■ady. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what

comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 4

```
simple_list = [0, 5, 2, 10, -1, 3]
```

Teacher Script:

Here we have Code Example 4. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 5

```
new_list = [] for elem in simple_list: new_list.append(2 * elem)
```

Teacher Script:

Here we have Code Example 5. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 6

```
new_list
```

Teacher Script:

Here we have Code Example 6. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 7

Now let's talk about Section 7. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 8

```
[2 * elem for elem in simple_list]
```

Teacher Script:

Here we have Code Example 8. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 9

Now let's talk about Section 9. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 10

```
div_by_3 =[n for n in range(1, 21) if n % 3 == 0]
```

Teacher Script:

Here we have Code Example 10. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations.

Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 11

Now let's talk about Section 11. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

If we also wanted to filter out elements greater than 0:

Now let's talk about **If we also wanted to filter out elements greater than 0:**. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 13

```
new_list = [] for elem in simple_list: if elem > 0: new_list.append(2 * elem)
new_list
```

Teacher Script:

Here we have Code Example 13. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 14

Now let's talk about Section 14. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 15

```
[2 * elem for elem in simple_list if elem > 0]
```

Teacher Script:

Here we have Code Example 15. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 16

Now let's talk about Section 16. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 17

```
[2 * elem if elem > 0 else elem for elem in simple_list]
```

Teacher Script:

Here we have Code Example 17. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations.

Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 18

```
dna = 'ACCGTA' alkali = ['puryna' if a == 'A' or a == 'G' else 'pirymidyna' for a in dna] alkali
```

Teacher Script:

Here we have Code Example 18. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 19

Now let's talk about Section 19. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Exercise

Now let's talk about Exercise. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 21

```
numbers = [2, 5, 3, 9, 1] # Your turn ■ # squares= [.....]
```

Teacher Script:

Here we have Code Example 21. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make

sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 23

Now let's talk about Section 23. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 24

```
tech = ["airflow", "dag", "spark", "sql"] # Your turn ■ # short = [...]
```

Teacher Script:

Here we have Code Example 24. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations.

Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 26

Now let's talk about Section 26. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?

A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

From a list of dictionaries , get the list of names

Now let's talk about **From a list of dictionaries , get the list of names**. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 28

```
kids = [ {"name": "Alan", "age": 3}, {"name": "Jessica", "age": 6}, {"name":
"Bryan", "age": 11} ] names = [kid["name"] for kid in kids] names
```

Teacher Script:

Here we have Code Example 28. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 29

Now let's talk about Section 29. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 30

```
ages_above_3 = [kid["age"] for kid in kids if kid["age"] > 3] ages_above_3
```

Teacher Script:

Here we have Code Example 30. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 31

Now let's talk about Section 31. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 32

```
products = [ {"id": 1, "name": "Laptop", "price": 3200, "discount": True}, {"id": 2, "name": "Mouse", "price": 150, "discount": False}, {"id": 3, "name": "Monitor", "price": 1200, "discount": True}, {"id": 4, "name": "Keyboard", "price": 400, "discount": False} ] discounted = [ {"name": product["name"], "discounted_price": round(product["price"] * 0.9, 2)} for product in products if product["discount"] ] discounted
```

Teacher Script:

Here we have Code Example 32. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 33

Now let's talk about Section 33. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 34

```
users = [ {"name":"Ala","email":"a@example.com","active":True,"country":"PL"},
{"name":"Bob","email":"b@example.com","active":False,"country":"PL"},
{"name":"Carl","email":"c@example.com","active":True,"country":"DE"},
{"name":"Dana","email":"d@example.com","active":True,"country":"PL"}, ] # Your
turn ■ # Place for solution # emails = [....]
```

Teacher Script:

Here we have Code Example 34. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?

A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 36

Now let's talk about Section 36. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 37

```
stock_levels = [ {"item": "Samsung", "price": 700.0, "quantity": 5, "currency":
"EUR"}, {"item": "IPhone", "price": 1000.0, "quantity": 3, "currency": "EUR"}, ]
```

Teacher Script:

Here we have Code Example 37. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 38

Now let's talk about Section 38. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 39

```
exchange_rate = 4.26
```

Teacher Script:

Here we have Code Example 39. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 40

Now let's talk about Section 40. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 41

```
stock_levels_pln = [ {**stock, **{"price": stock["price"] * exchange_rate,
"currency": "PLN"}} for stock in stock_levels ] stock_levels_pln
```

Teacher Script:

Here we have Code Example 41. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 42

Now let's talk about Section 42. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?

A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 43

```
transactions = [ {"id": 1, "item": "TV", "quantity": 2, "price": 2000.0}, {"id": 2, "item": "TV", "quantity": 1, "price": 4000.0}, {"id": 3, "item": "PC", "quantity": 3, "price": 2500.0}, ]
```

Teacher Script:

Here we have Code Example 43. First, let’s look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it’s the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 44

```
# Your turn ■ # Place for solution
```

Teacher Script:

Here we have Code Example 44. First, let’s look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it’s the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let’s talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they’ve seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What’s the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 46

Now let’s talk about Section 46. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they’ve seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What’s the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 47

```
data = [None, "12", -5, ("result", 8), "7", None, 5, "-8", ("ranking", 15), 0, None, "3", -2, ("value", 21)] result = [0, 3, 5, 7, 8, 12, 15, 21]
```

Teacher Script:

Here we have Code Example 47. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 48

```
# Your turn ■ # HINT # isinstance() is a built-in Python function used to check the type of an object. # isinstance(object, type) # x = 5 # isinstance(x, int) # → True # Place for solution
```

Teacher Script:

Here we have Code Example 48. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 50

Now let's talk about Section 50. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 51

Now let's talk about Section 51. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 52

```
temperatures = { "Pozna■": 10, "Warszawa": 5, "Kraków": 8, "Wroc■aw": 12,
"Gda■sk": 7, } temperatures
```

Teacher Script:

Here we have Code Example 52. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 53

Now let's talk about Section 53. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 54

```
temperatures.items()
```

Teacher Script:

Here we have Code Example 54. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 55

```
for day, temp in temperatures.items(): print(day, temp)
```

Teacher Script:

Here we have Code Example 55. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 56

```
temperatures.keys()
```

Teacher Script:

Here we have Code Example 56. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 57

```
temperatures.values()
```

Teacher Script:

Here we have Code Example 57. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 58

Now let's talk about Section 58. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 59

```
temperatures_dict = {} for city, temp_ in temperatures.items(): if temp_ > 7:
temperatures_dict[city] = temp_ temperatures_dict
```

Teacher Script:

Here we have Code Example 59. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 60

Now let's talk about Section 60. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 61

```
{ city: temp_ for city, temp_ in temperatures.items() if temp_ > 7}
```

Teacher Script:

Here we have Code Example 61. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 62

Now let's talk about Section 62. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 63

```
def celsius2fahrenheit(c): return (c * 9/5) + 32
```

Teacher Script:

Here we have Code Example 63. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 64

Now let's talk about Section 64. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 65

```
{ city: celsius2fahrenheit(temp_) for city, temp_ in temperatures.items() if
temp_ > 7 }
```

Teacher Script:

Here we have Code Example 65. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 66

Now let's talk about Section 66. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 67

```
temperatures_list = [(city, celsius2fahrenheit(temp_)) for city, temp_ in
temperatures.items() if temp_ > 7] temperatures_list
```

Teacher Script:

Here we have Code Example 67. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 68

Now let's talk about Section 68. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?

A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 69

```
{ city: temp_ for city, temp_ in temperatures_list }
```

Teacher Script:

Here we have Code Example 69. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 70

Now let's talk about Section 70. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 71

```
dict(temperatures_list)
```

Teacher Script:

Here we have Code Example 71. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 72

Now let's talk about Section 72. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 73

```
data = [ {"id": 1, "name": "Alicja", "salary": 7000}, {"id": 2, "name": "Robert",
"salary": 5000}, {"id": 3, "name": "Jerzy", "salary": 12000} ]
```

Teacher Script:

Here we have Code Example 73. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 74

```
# Your turn ■ # Place for solution
```

Teacher Script:

Here we have Code Example 74. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 76

Now let's talk about Section 76. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 77

```
data = [ {"id": 1, "name": "Alicja", "salary": 7000}, {"id": 2, "name": "Robert",
"salary": 5000}, {"id": 3, "name": "Jerzy", "salary": 12000}, {"id": 4, "name":
"Alicja", "salary": 9999}, ]
```

Teacher Script:

Here we have Code Example 77. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 78

Now let's talk about Section 78. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 79

```
{ d["name"] for d in data if d["salary"] < 10000 }
```

Teacher Script:

Here we have Code Example 79. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 80

Now let's talk about Section 80. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 81

```
{ d["name"]: d["salary"] for d in data if d["salary"] < 10000 }
```

Teacher Script:

Here we have Code Example 81. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 82

Now let's talk about Section 82. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Let's use an example to analyze temperature measurements. Each row represents a separate measurement device. The values of the elements in the rows represent subsequent measurements.

Now let’s talk about **Let's use an example to analyze temperature measurements. Each row represents a separate measurement device. The values of the elements in the rows represent subsequent measurements.**. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they’ve seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What’s the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 84

```
temperatures = [ [-10, 30, 8], [-50, 0, -20, 10], [-30, 25], [5, 40, 22], ]
```

Teacher Script:

Here we have Code Example 84. First, let’s look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it’s the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 85

Now let's talk about Section 85. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 86

```
def celsius2fahrenheit(c): return (c * 9/5) + 32
```

Teacher Script:

Here we have Code Example 86. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 87

```
result = [] for internal_temp in temperatures:
result.append([celsius2fahrenheit(element) for element in internal_temp if
element > 0]) result
```

Teacher Script:

Here we have Code Example 87. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 88

Now let's talk about Section 88. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 89

```
[ [celsius2fahrenheit(element) for element in wewn_temp if element > 0] for
wewn_temp in temperatures ]
```

Teacher Script:

Here we have Code Example 89. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 90

Now let's talk about Section 90. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 91

```
temp_list = [] for internal_temp in temperatures: temp_list +=
[celsius2fahrenheit(element) for element in internal_temp if element > 0]
temp_list
```

Teacher Script:

Here we have Code Example 91. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?

A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 92

Now let's talk about Section 92. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 93

```
'''temperatures = [ [-10, 30, 8], [-50, 0, -20, 10], [-30, 25], [5, 40, 22], ]'''
[ celsius2fahrenheit(element) for internal_temp in temperatures for element in
internal_temp if element > 0 ]
```

Teacher Script:

Here we have Code Example 93. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 94

Now let's talk about Section 94. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of

preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 95

```
products = [ ["Apple", "Banana", "Avocado"], ["Apricot", "Blueberry"], ["Almond", "Cherry", "Anise"] ] # Your turn ■ # Place for solution
```

Teacher Script:

Here we have Code Example 95. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 97

Now let's talk about Section 97. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Example

Now let's talk about Example. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?

A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 99

```
grades = { "Alice": {"Math": 85, "Science": 90}, "Bob": {"Math": 55, "Science": 75}, "Charlie": {"Math": 92, "Science": 88} } passed_math = {student: subjects for student, subjects in grades.items() if subjects["Math"] >= 60} passed_math
```

Teacher Script:

Here we have Code Example 99. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 100

Now let's talk about Section 100. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 101

```
logs = { "server_1": { "service_a": {"cpu": 55, "memory": 1200, "requests":
3200}, "service_b": {"cpu": 70, "memory": 1800, "requests": 4500} }, "server_2":
{ "service_a": {"cpu": 65, "memory": 1400, "requests": 3800}, "service_c":
{"cpu": 80, "memory": 2200, "requests": 5100} } }
```

Teacher Script:

Here we have Code Example 101. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 102

```
# Your turn ■ # Placeholder for solution - variant I
```

Teacher Script:

Here we have Code Example 102. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 104

```
# Your turn ■ # Placeholder for solution - variant II
```

Teacher Script:

Here we have Code Example 104. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?

A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 106

Now let's talk about Section 106. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 107

Now let's talk about Section 107. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?

A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 108

```
def countdown(n: int): print("Start!") while n > 0: print("About to yield", n) yield n n -= 1 print("Done.") gen = countdown(10) # doesn't compute anything yet gen
```

Teacher Script:

Here we have Code Example 108. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 109

Now let's talk about Section 109. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 110

```
next(gen) # consumes one value at a time / pobiera jedna warto■■ i zatrzymuje si■
```

Teacher Script:

Here we have Code Example 110. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 111

```
list(gen) # continue consuming remaining values / pobiera reszt■ warto■ci
```

Teacher Script:

Here we have Code Example 111. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 112

Now let's talk about Section 112. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 113

```
generator_ = (x**2 for x in range(10))
```

Teacher Script:

Here we have Code Example 113. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 114

```
print(type(generator_))
```

Teacher Script:

Here we have Code Example 114. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 115

```
list = [x**2 for x in range(10)]
```

Teacher Script:

Here we have Code Example 115. First, let’s look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it’s the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Let's check if the result is as expected:

Now let’s talk about **Let's check if the result is as expected:**. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they’ve seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What’s the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 117

```
list(generator_)
```

Teacher Script:

Here we have Code Example 117. First, let’s look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it’s the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 118

Now let’s talk about Section 118. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they’ve seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What’s the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Generators can be used to create call chains that resemble data flows. This way, instead of triggering calculations for subsequent variables, we create a "recipe" for obtaining them. For example: (lazy, memory-efficient pipelines)

Now let’s talk about **Generators can be used to create call chains that resemble data flows. This way, instead of triggering calculations for subsequent variables, we create a "recipe" for obtaining them. For example: (lazy, memory-efficient pipelines)**. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they’ve seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What’s the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 120

```
squares = (x**2 for x in range(10))
```

Teacher Script:

Here we have Code Example 120. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 121

```
divided_by_3 = (x for x in squares if x % 3 == 0)
```

Teacher Script:

Here we have Code Example 121. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

At this point, the calculations are performed:

Now let's talk about **At this point, the calculations are performed:**. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe.

This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 123

```
list(divided_by_3)
```

Teacher Script:

Here we have Code Example 123. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 124

Now let's talk about Section 124. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?

A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 125

Now let's talk about Section 125. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Exercise

Now let's talk about Exercise. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 127

```
logs = [ "2025-08-26 INFO User logged in", "2025-08-26 ERROR Connection lost", "2025-08-26 INFO Data saved", "2025-08-26 ERROR Timeout", ] # Your turn ■ # Place for solution # errors = (.....)
```

Teacher Script:

Here we have Code Example 127. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Section 129

Now let's talk about Section 129. List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Code Example 130

```
# Dataset (CSV-like rows) csv_rows = [ "1,TV,2000", "2,Laptop,3500",
"3,Phone,1500", ] # def iter_prices(lines): # for row in lines: # _, _, price =
row.strip().split(",") # yield float(price) # Your turn ■ # Place for solution -
modify function and teste whether it works
```

Teacher Script:

Here we have Code Example 130. First, let's look at the input data. Then we walk through the loop step by step. Notice how in the traditional for loop we create an empty list, append elements one by one, and then print the result. (pause here) Now compare this with the list comprehension — it's the same logic, but much shorter and more elegant. Before running it, ask participants to predict the output. After running, confirm their expectations. Emphasize that both approaches are correct, but list comprehensions are often preferred.

Possible Questions & Answers:

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.

Now let's talk about . List comprehensions are a way to create lists in Python in a single line of code. They make the code shorter, cleaner, and often faster. (pause here) Think of it like cooking: instead of preparing each ingredient separately, you combine them all in one recipe. This is exactly what comprehensions do with data. They are very common in real-world Python projects, especially in data engineering and analysis. Make sure everyone understands that readability is just as important as speed. (ask participants if they've seen list comprehensions before).

Possible Questions & Answers:

Q: Why are list comprehensions preferred?
A: They improve readability, conciseness, and are often faster.

Q: When should I avoid list comprehensions?
A: Avoid them when the logic is too complex or readability would suffer.

Q: What's the difference between list comprehensions and generator expressions?
A: Generator expressions use parentheses and are lazy (generate items one by one). List comprehensions create the full list in memory immediately.

Q: Can I use if-else inside a comprehension?
A: Yes. Example: [x if x % 2 == 0 else -x for x in range(5)].

Q: Can list comprehensions be nested?
A: Yes, though it may hurt readability. Example: [[x*y for y in range(3)] for x in range(2)].

Q: How do comprehensions relate to functional tools like map/filter?
A: They are more Pythonic, often replacing map/filter with a clearer syntax.