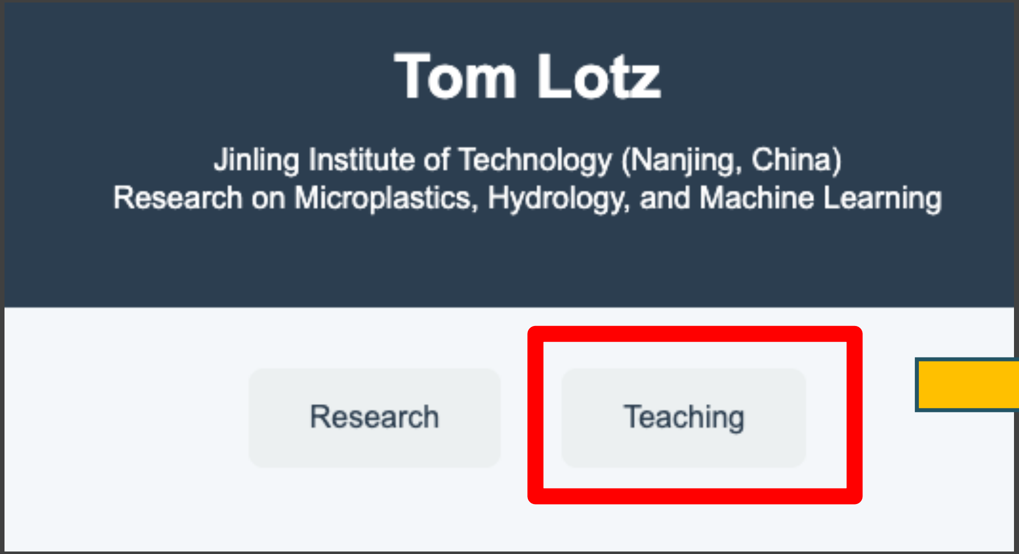
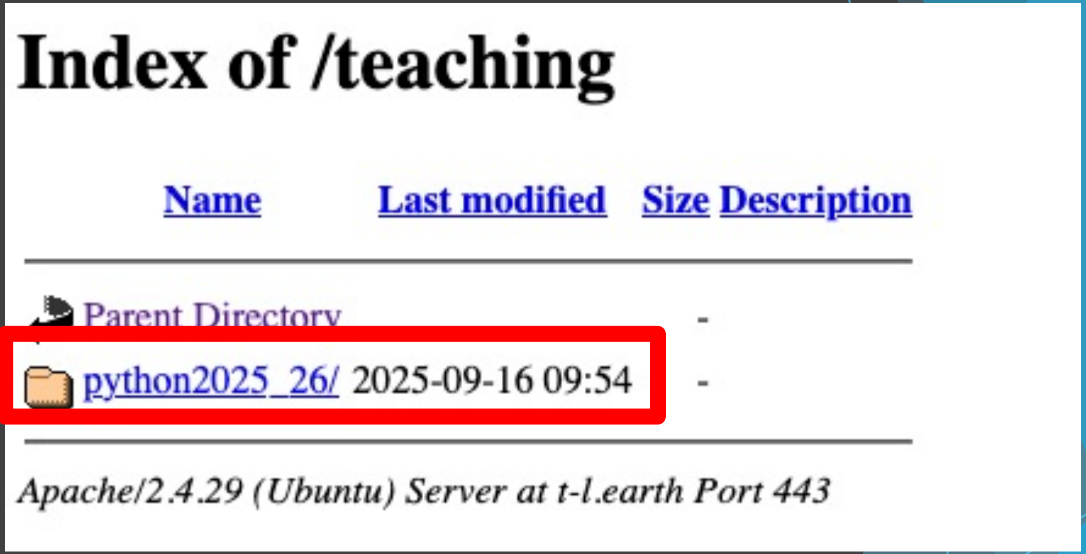
Download the cheat sheets and slides from here

t-l.earth



Tom Lotz

Jinling Institute of Technology (Nanjing, China)
Research on Microplastics, Hydrology, and Machine Learning



Index of /teaching

Name	Last modified	Size	Description
Parent Directory		-	
python2025_26/	2025-09-16 09:54	-	

Apache/2.4.29 (Ubuntu) Server at t-l.earth Port 443

Python语言程序设计

Python Programming

2025/26

Session 02

Tom Lotz (tom.lotz@outlook.com)

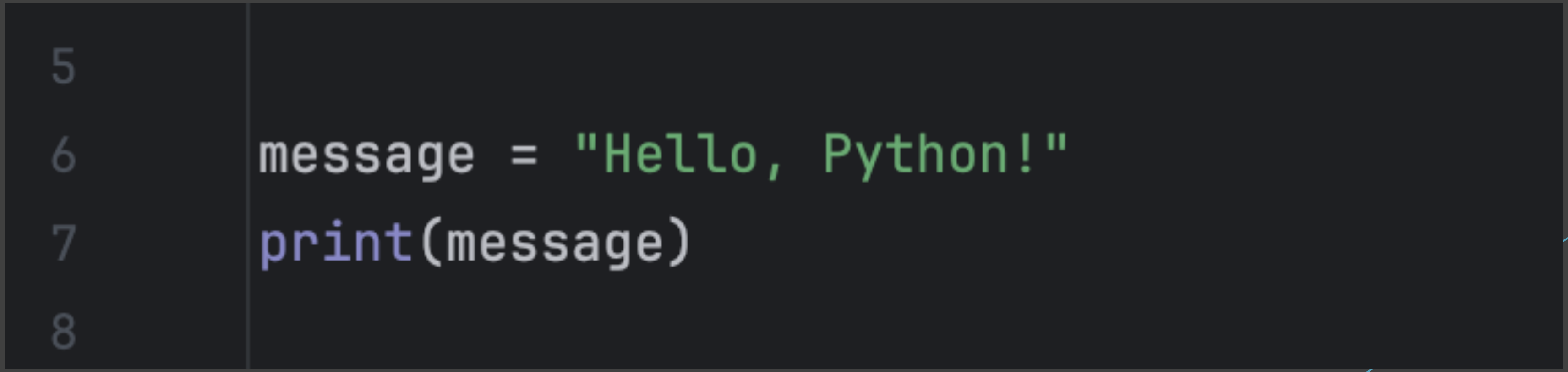
Content

Review

Using a Variable

REVIEW!

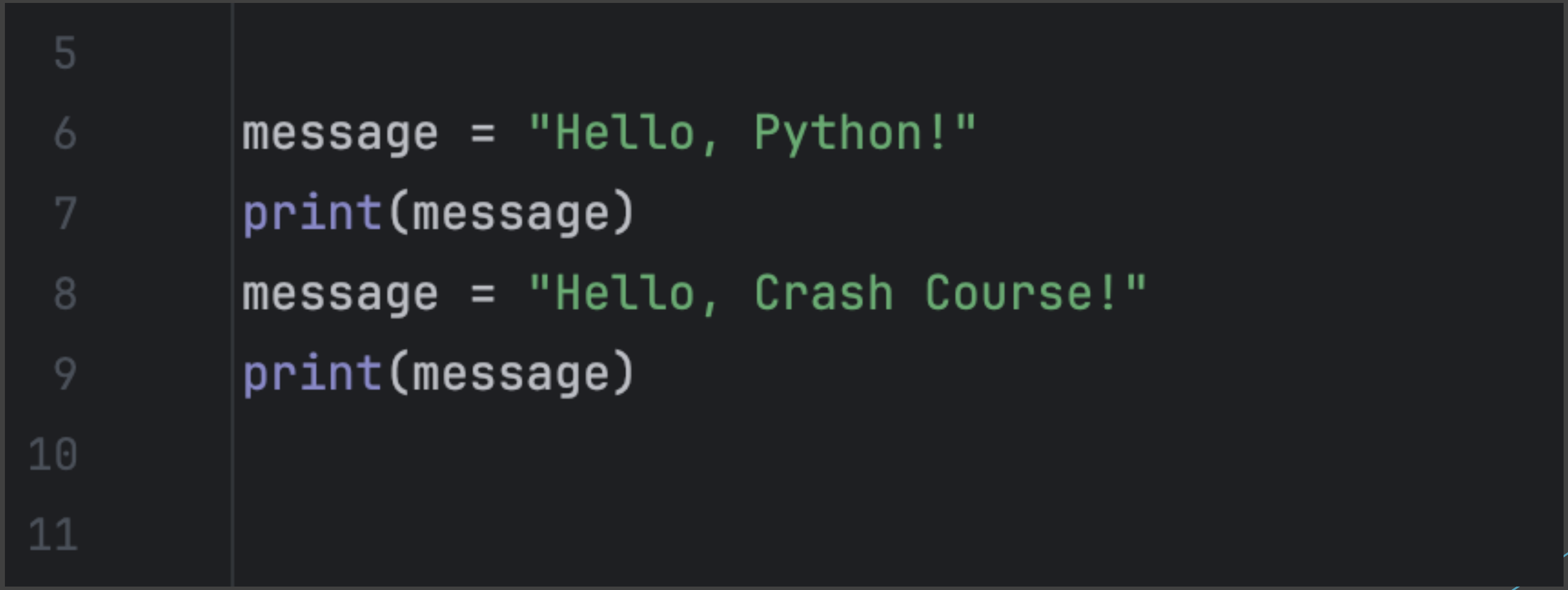
- A variable stores a value (like a label)
- Variable declaration is very easy in Python – just assign a value
- You can reuse and change it
- message is the variable name



```

message = "Hello, Python!"
print(message)

```

Using a Variable

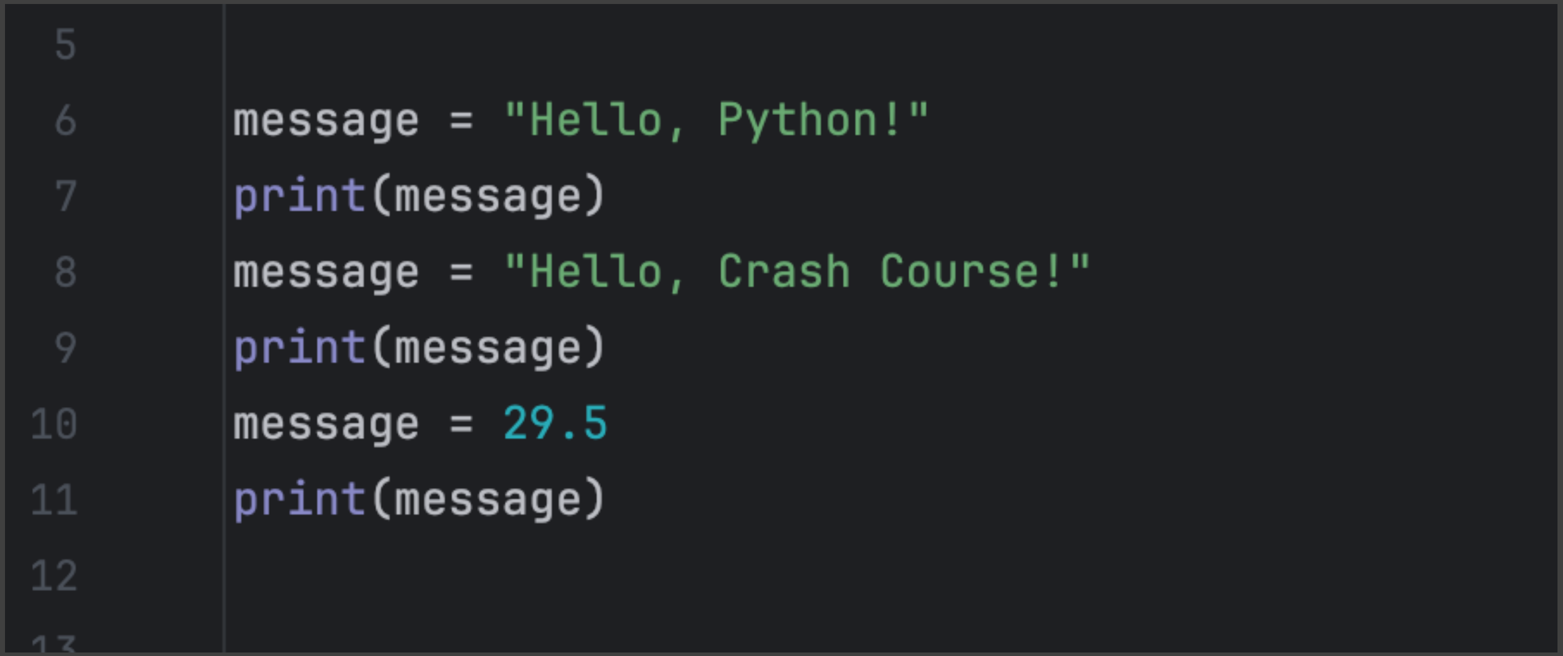
- You can assign a new value to a variable anytime
- Python will always use the latest value



```python

message = "Hello, Python!"
print(message)
message = "Hello, Crash Course!"
print(message)


```

Careful!

- This flexibility of Python can cause problems

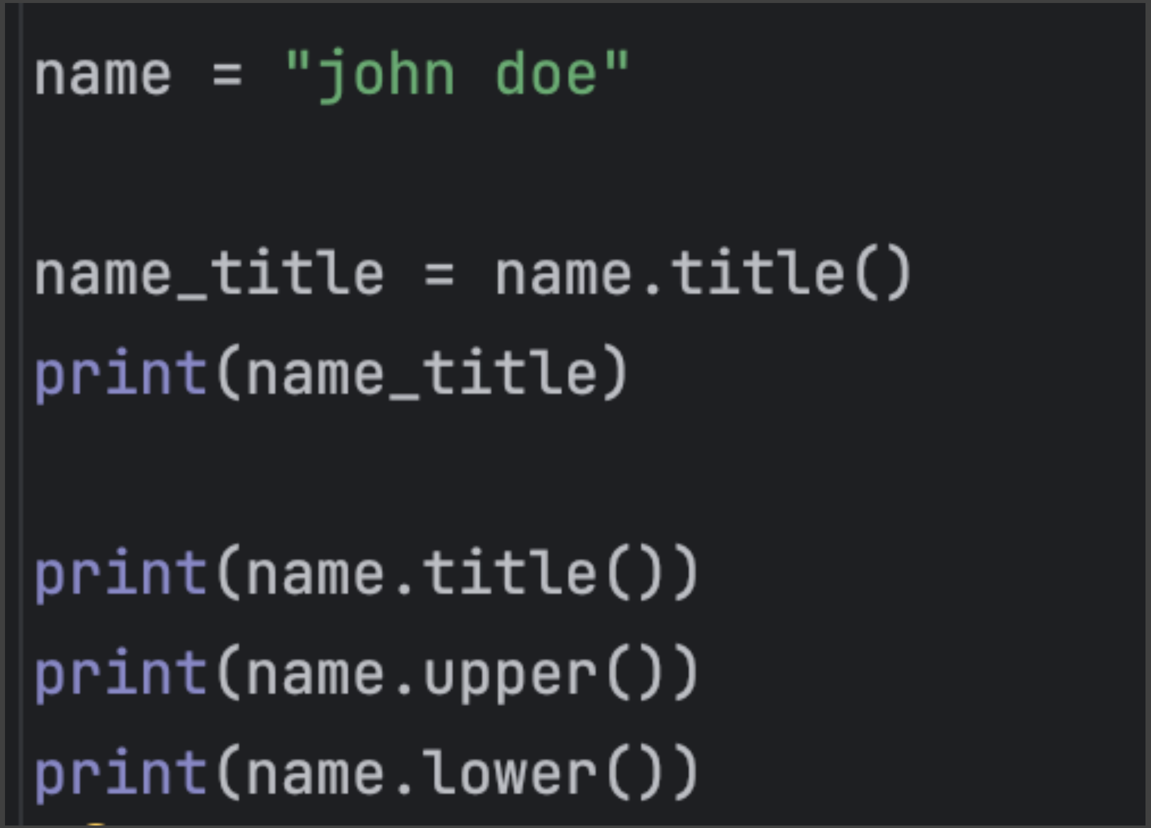


```python
message = "Hello, Python!"
print(message)
message = "Hello, Crash Course!"
print(message)
message = 29.5
print(message)
```

Variable Naming Rules

- Use letters, numbers, and underscores: `greeting_1`
- Must start with a letter or underscore (not a number)
- No spaces allowed
- Cannot use Python keywords (like `print`, `for`, etc.)
- Use lowercase and descriptive names: `user_name`, not `u`

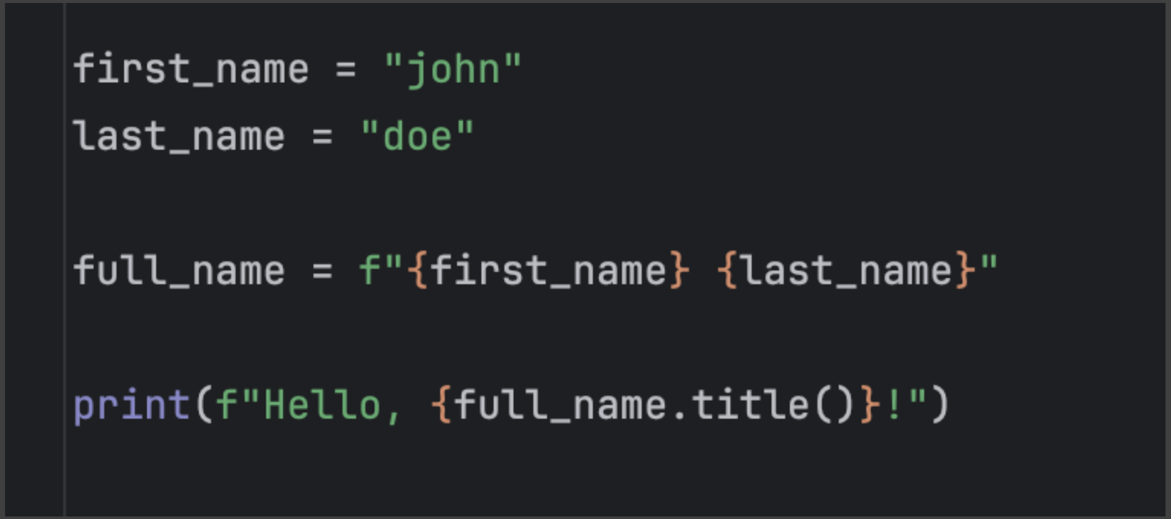
Strings and String Methods



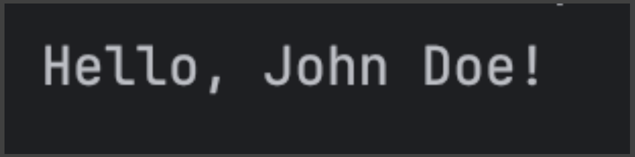
```
name = "john doe"

name_title = name.title()
print(name_title)

print(name.title())
print(name.upper())
print(name.lower())
```

f-Strings (Formatted Strings)

- f-strings let you insert variables inside strings
- Very useful for dynamic messages

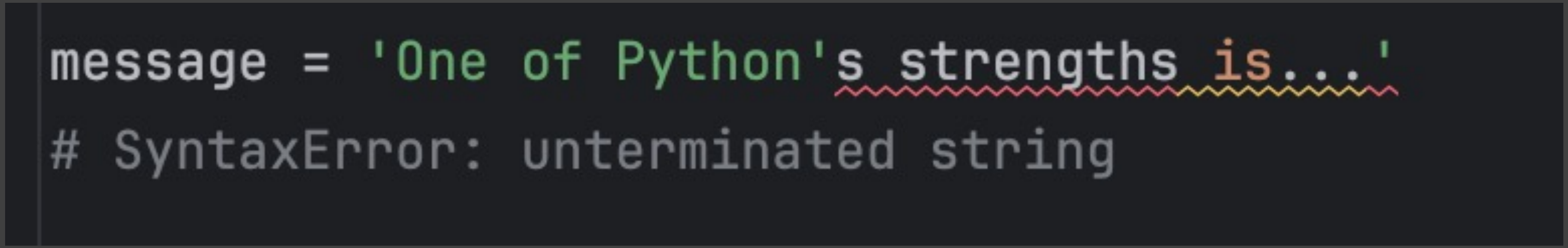


```python
first_name = "john"
last_name = "doe"

full_name = f"{first_name} {last_name}"

print(f"Hello, {full_name.title()}!")
```

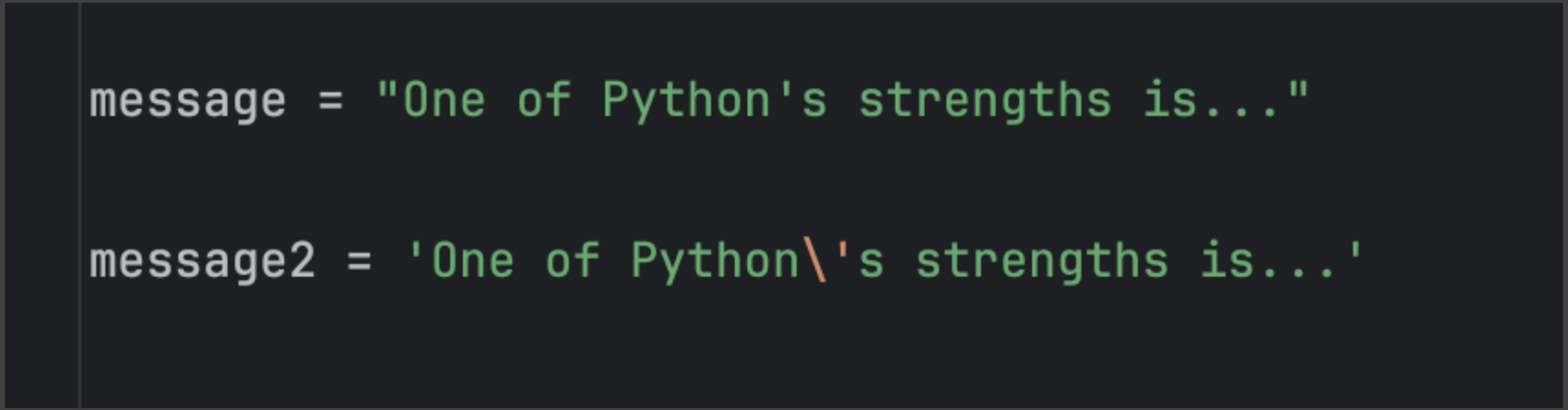


```
Hello, John Doe!
```

Syntax Errors with Strings



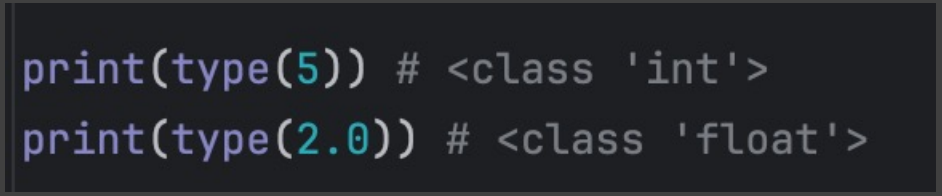
```python
message = 'One of Python's strengths is...'
# SyntaxError: unterminated string
```



```python
message = "One of Python's strengths is..."

message2 = 'One of Python\'s strengths is...'
```

Python Number Types

- **Integers** (whole numbers)
  - Examples: -2, 0, 42
- **Floats** (numbers with decimals)
  - Examples: 3.14, 0.0, -2.5
- Python automatically chooses the type



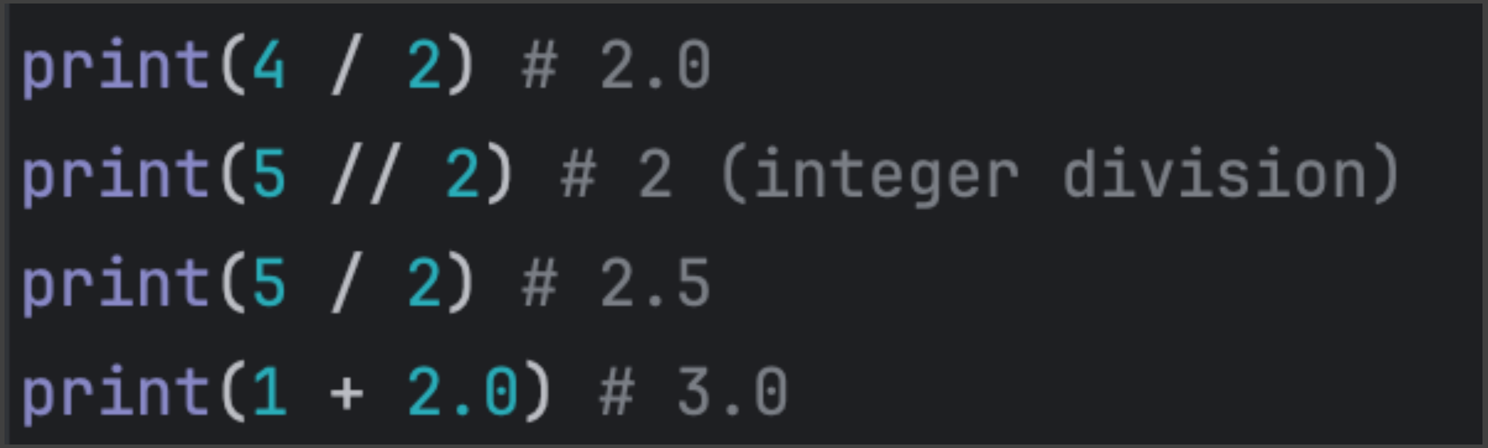
```
print(type(5)) # <class 'int'>
print(type(2.0)) # <class 'float'>
```

- You don’t need to declare types in advance

Division and Mixed Operations

REVIEW!

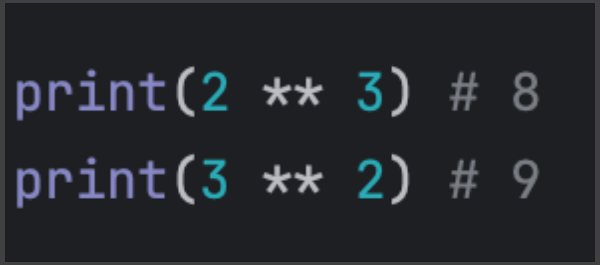
- `/` always returns a float, even for integers
- `//` performs integer division (truncates decimal)
- Mixing int and float in any operation → result is a float



```python
print(4 / 2) # 2.0
print(5 // 2) # 2 (integer division)
print(5 / 2) # 2.5
print(1 + 2.0) # 3.0
```

Exponents and Floats

- ** is the exponent operator



```python
print(2 ** 3) # 8
print(3 ** 2) # 9
```

Exercise

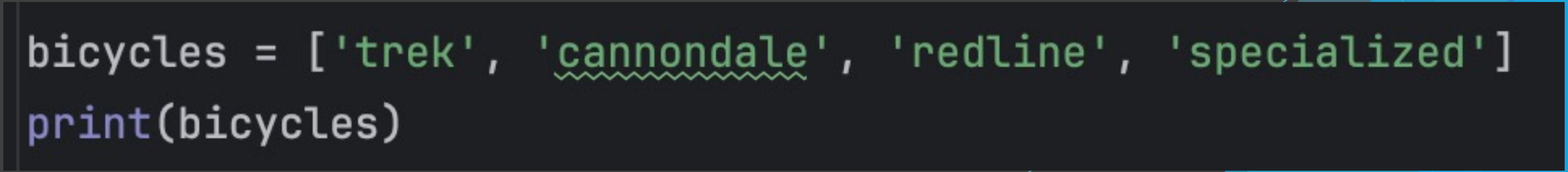
- *Clean up the string from string_session2.txt found at* *http://www.t-l.earth/teaching/python2025_26/special/* *using the string methods we have covered in Session 1.*

```
message = ""
```

Understanding and Creating Lists

What Is a List?

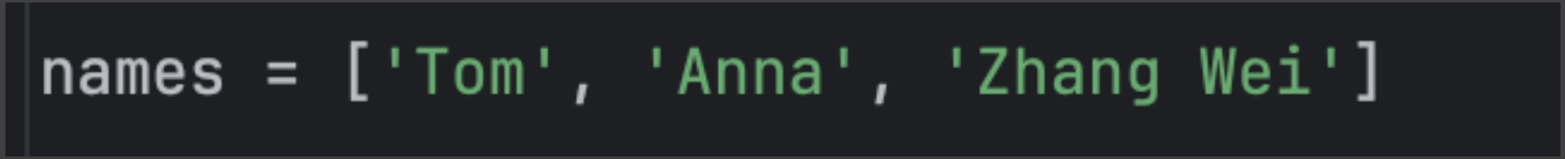
- A list is a collection of items in a particular order.
- Lists can contain:
    - Strings, numbers, booleans, other lists
    - Even mixed types (though not recommended)
- Think of a list as a shelf where each item has a position.



```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

List Syntax

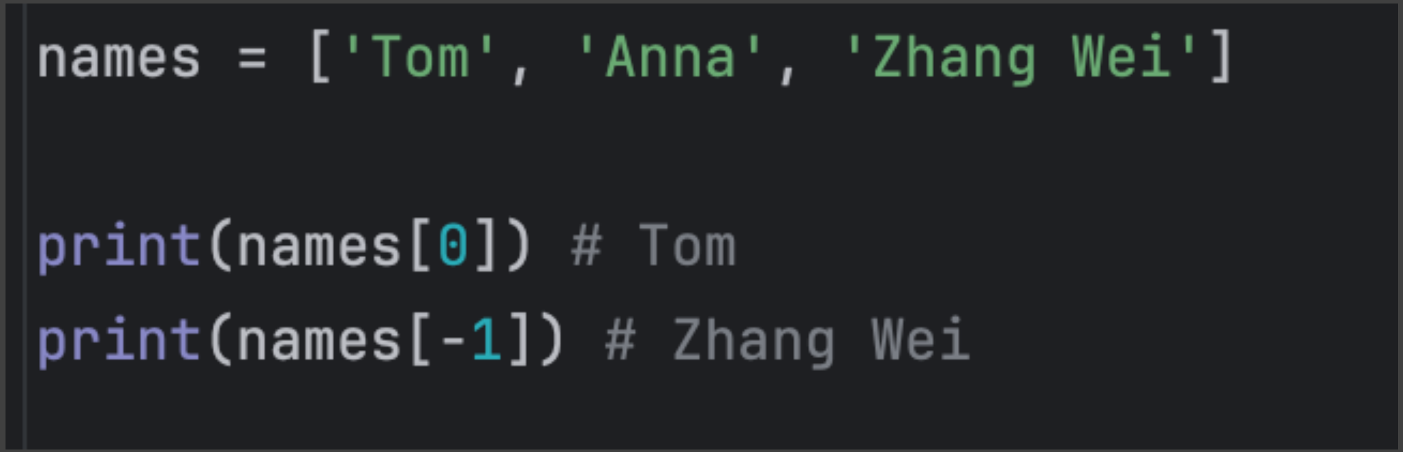
- Lists use square brackets []
- Items are separated by commas ,
- Choose plural names for variables (e.g., names, cars)



```python
names = ['Tom', 'Anna', 'Zhang Wei']
```

Accessing Elements by Index

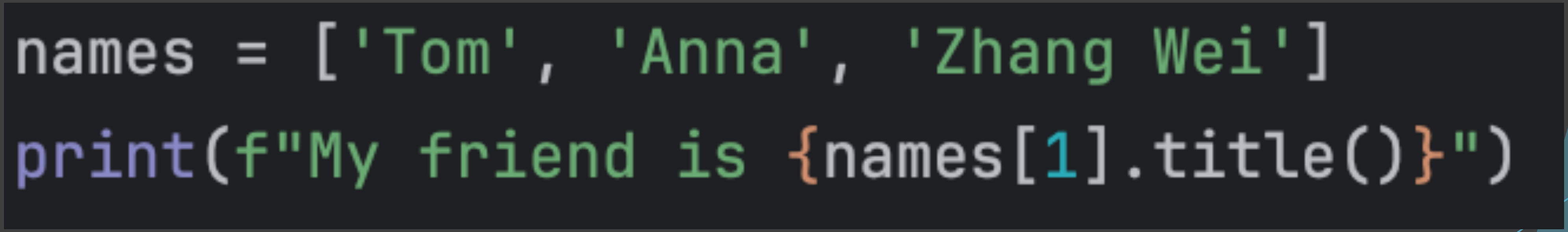
- Use brackets to get an item: list[index]
- Index starts at 0
- Negative indices count from the end



```
names = ['Tom', 'Anna', 'Zhang Wei']

print(names[0]) # Tom
print(names[-1]) # Zhang Wei
```

Formatting List Items

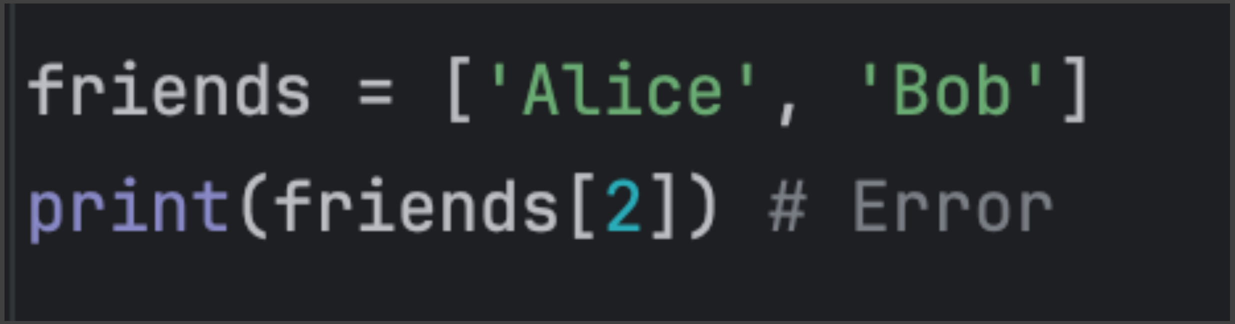
- You can use string methods like .title()
- Combine values with f-strings



```
names = ['Tom', 'Anna', 'Zhang Wei']
print(f"My friend is {names[1].title()}")
```

Common Pitfall: Index Errors

- IndexError: list index out of range
- Happens when you access a non-existent item



```python
friends = ['Alice', 'Bob']
print(friends[2]) # Error
```

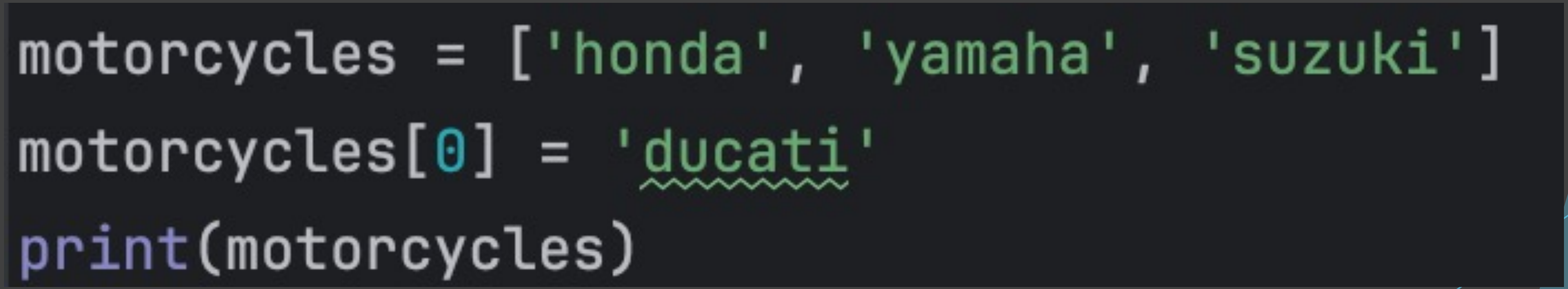
Key Takeaways

- Lists let you group related values
- You can access items with indexing
- Negative indices count from the end
- Lists work well with string formatting
- You will use lists everywhere in Python

Modifying Lists: Add, Change, Remove

Changing Elements in a List

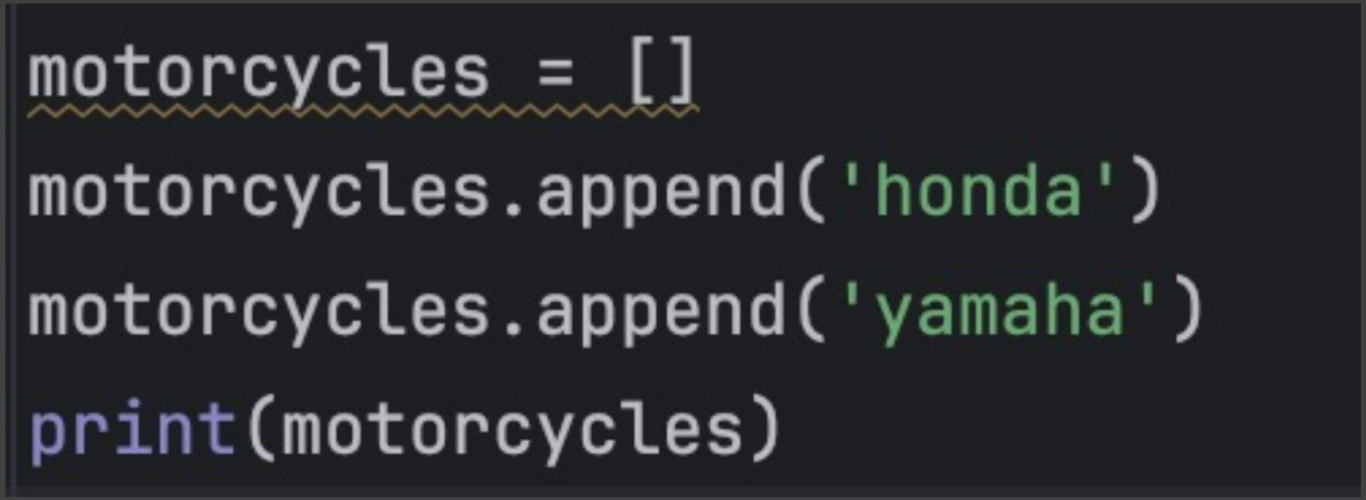
- You can change any value by accessing it via its index
- Syntax: list[index] = new_value



```python
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles[0] = 'ducati'
print(motorcycles)
```

Adding Items with append()

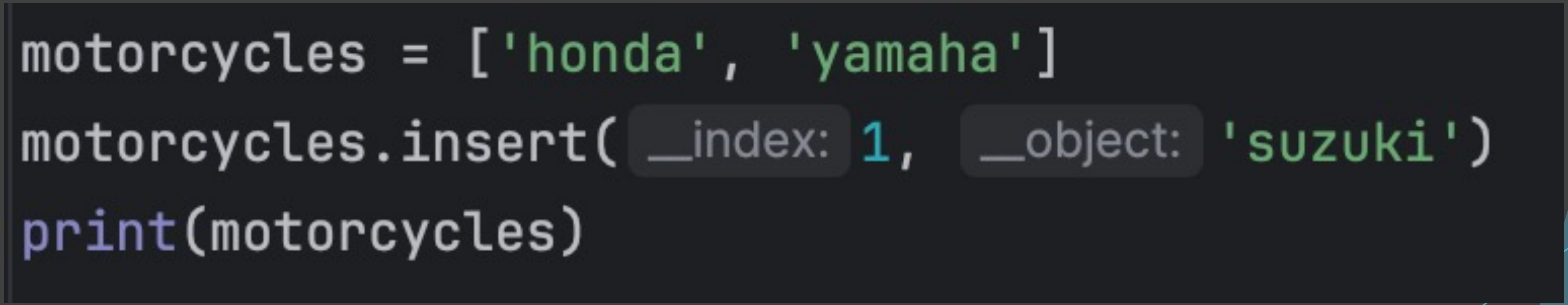
- append() adds an element to the end of the list
- Often used to build lists dynamically



```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
print(motorcycles)
```

Inserting Elements

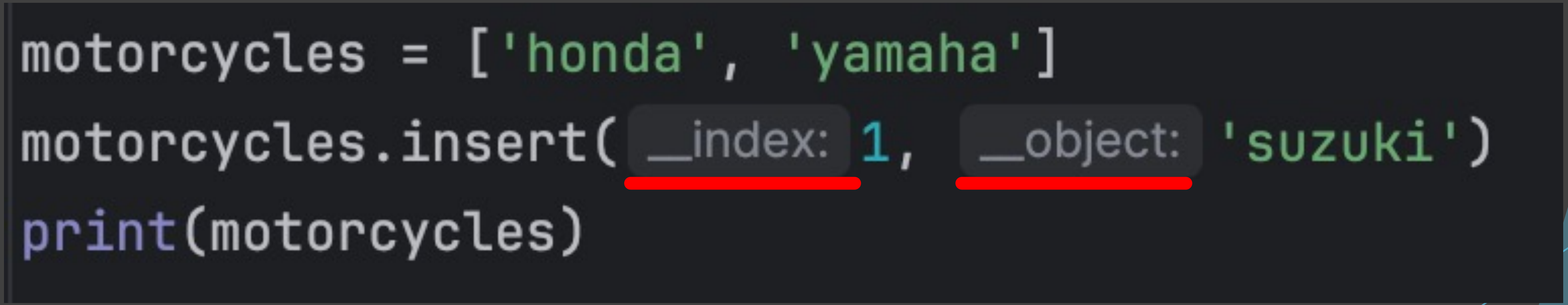
- insert(index, value) adds an item at any position
- Shifts following elements right



```
motorcycles = ['honda', 'yamaha']
motorcycles.insert( __index: 1,  __object: 'suzuki')
print(motorcycles)
```

Inserting Elements

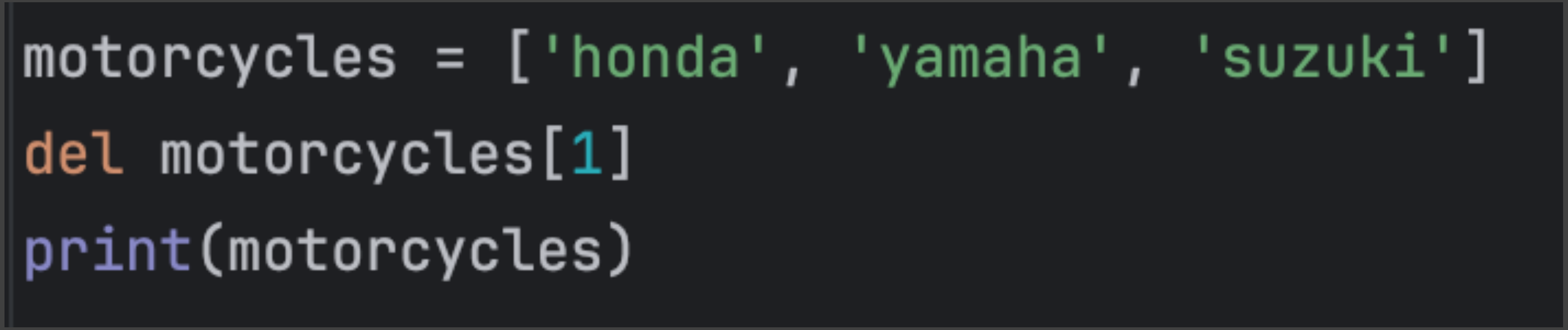
- insert(index, value) adds an item at any position
- Shifts following elements right



```
motorcycles = ['honda', 'yamaha']
motorcycles.insert( __index: 1,  __object: 'suzuki')
print(motorcycles)
```

Removing Items with del

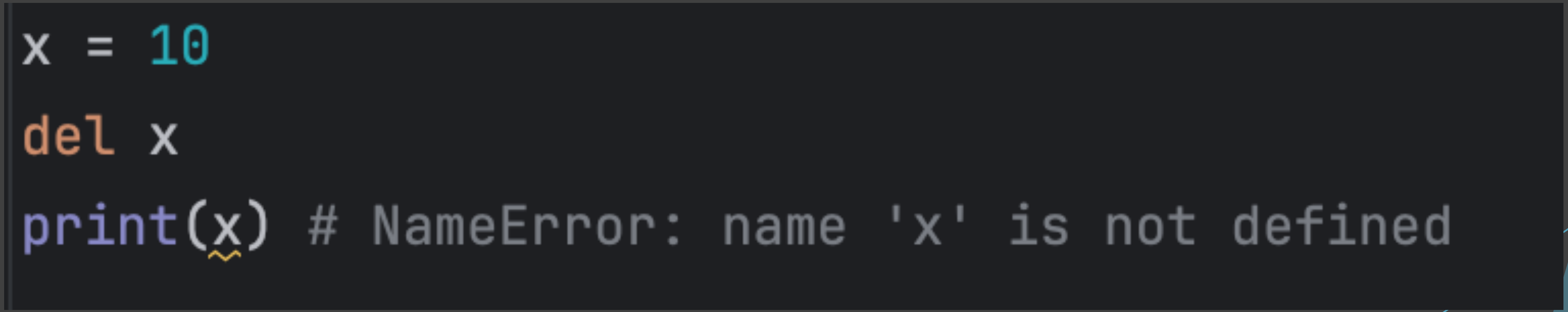
- del removes an item by index
- You cannot access the value after deleting



```python
motorcycles = ['honda', 'yamaha', 'suzuki']
del motorcycles[1]
print(motorcycles)
```

By the way: del ?

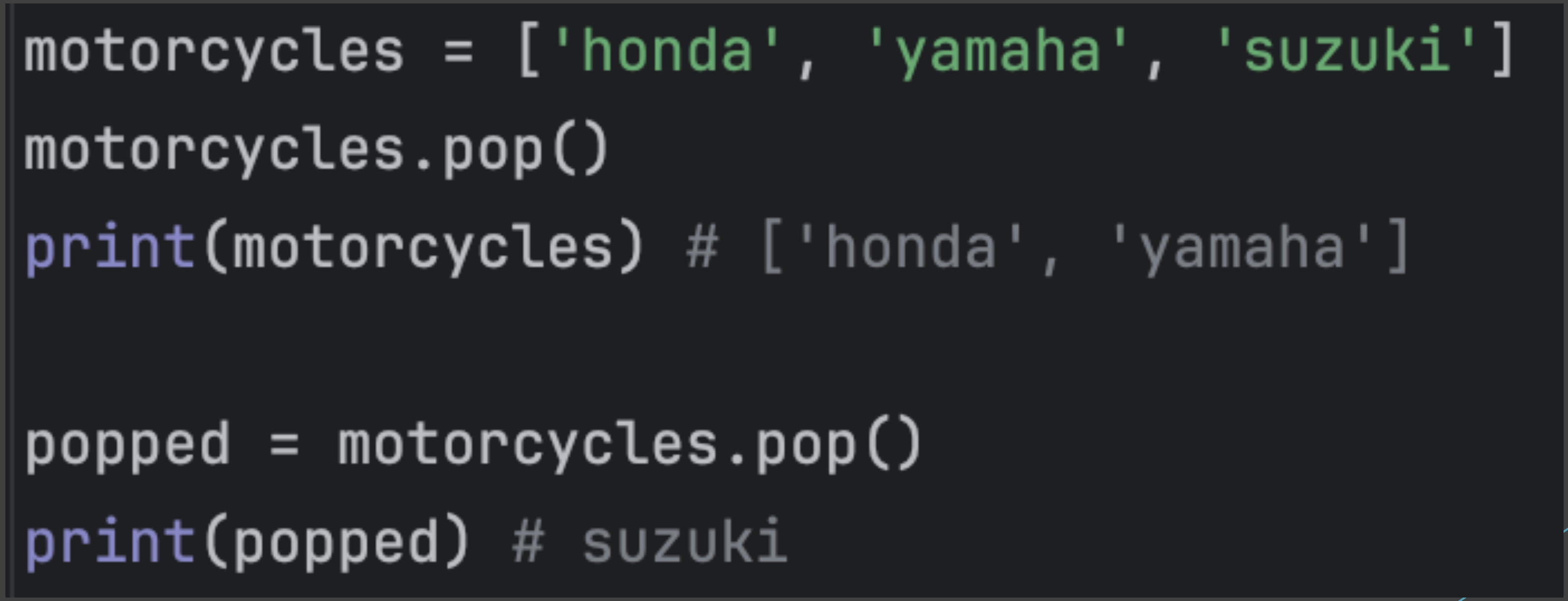
- del is a keyword in Python, it is not a function (similar to if, for, return)
- It is not specific to lists
- Can be used to delete many things



```
x = 10
del x
print(x) # NameError: name 'x' is not defined
```

Removing Items with pop()

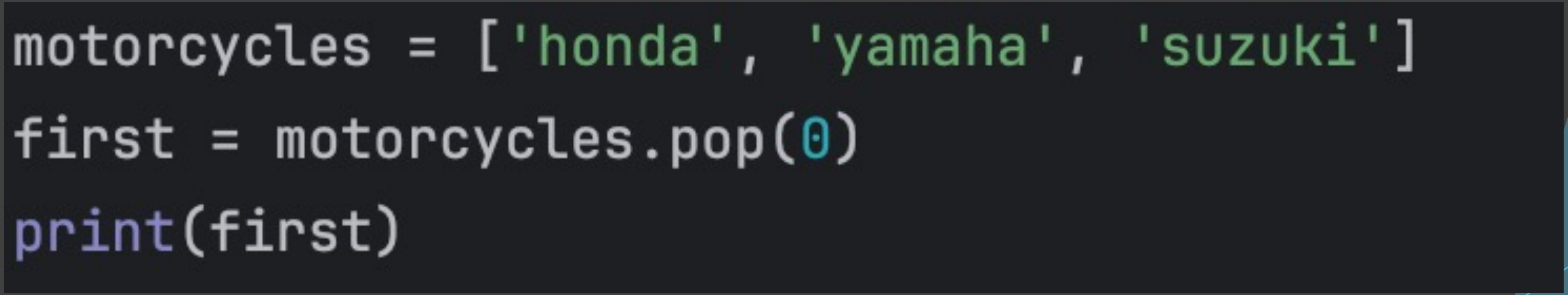
- pop() removes the last item by default
- You can store and use the removed item



```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.pop()
print(motorcycles) # ['honda', 'yamaha']

popped = motorcycles.pop()
print(popped) # suzuki
```

pop() from Specific Position

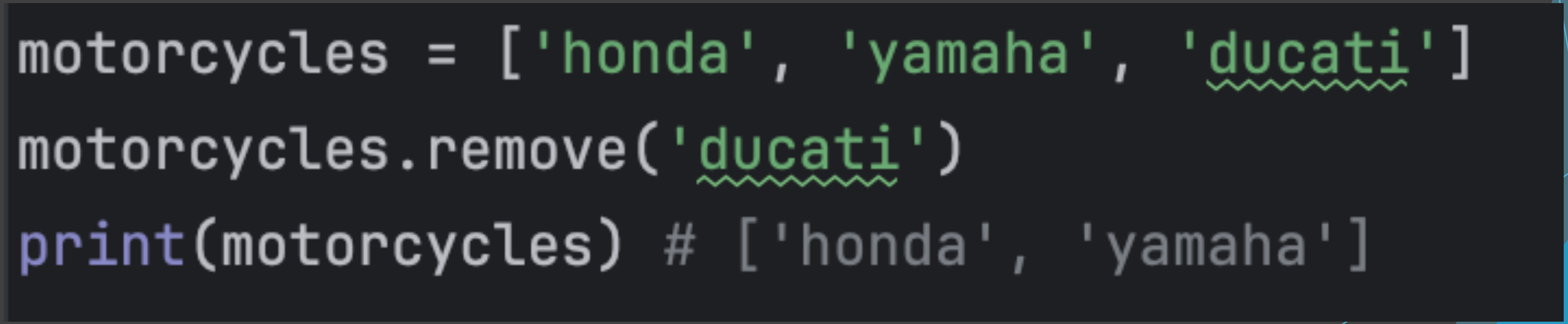
- You can specify the index inside pop(index)



```
motorcycles = ['honda', 'yamaha', 'suzuki']
first = motorcycles.pop(0)
print(first)
```

Removing by Value with remove()

- remove(value) deletes the first occurrence
- Useful when you know the value, not the index



```
motorcycles = ['honda', 'yamaha', 'ducati']
motorcycles.remove('ducati')
print(motorcycles) # ['honda', 'yamaha']
```

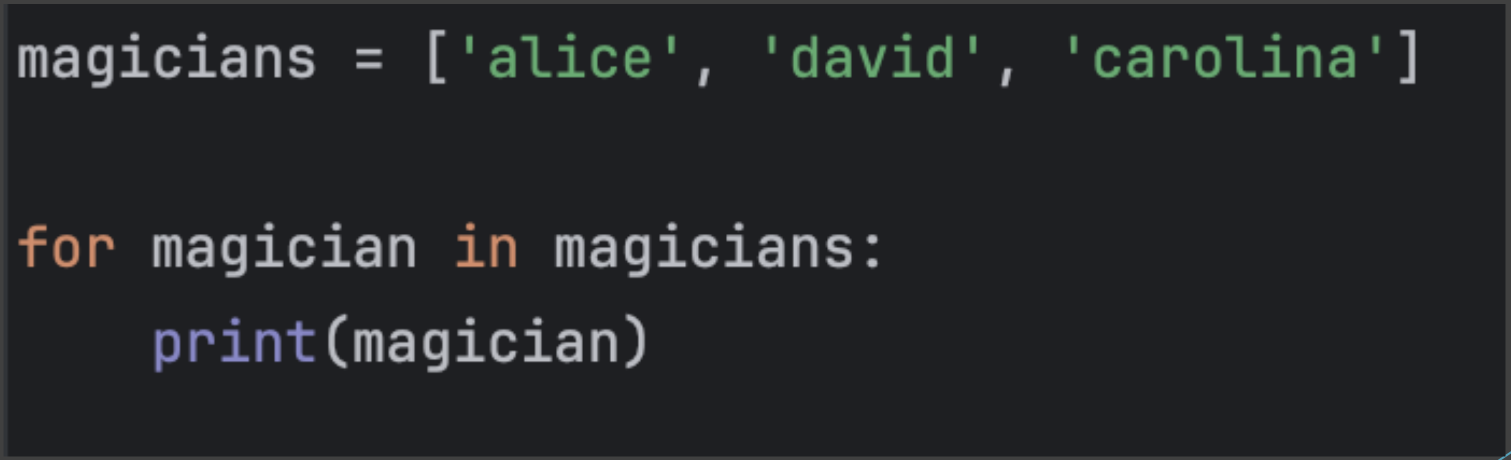
Key Takeaways

- Lists are dynamic: you can change, add, and remove items
- Use append(), insert(), del, pop(), and remove() depending on the need
- pop() and remove() let you keep the item for later use

Looping and Organizing Lists

Looping Through a List

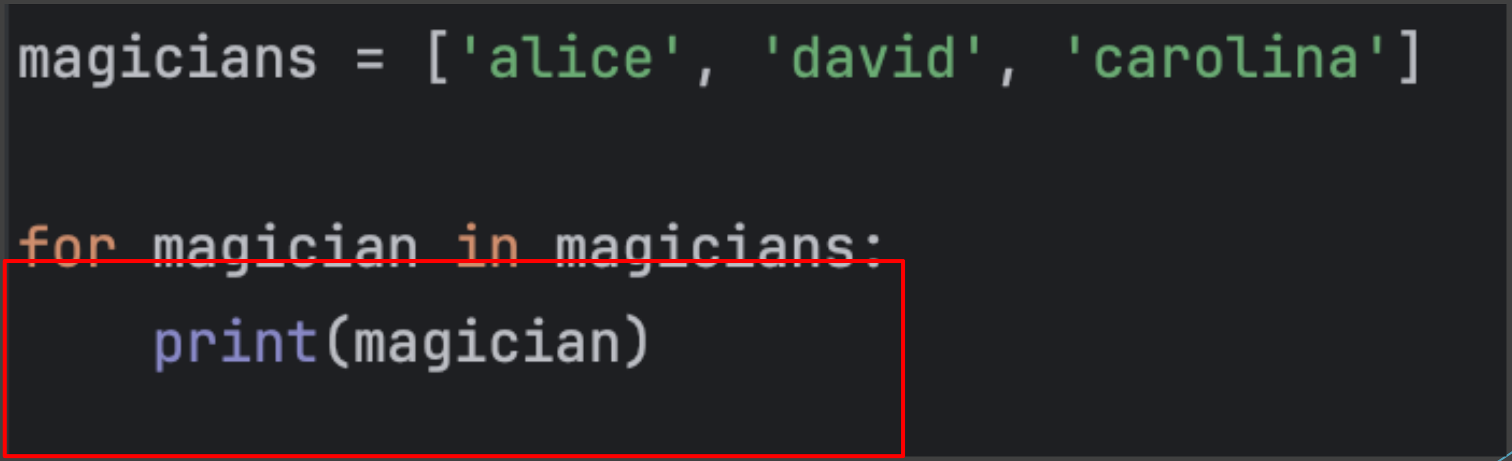
- Use a for loop to repeat actions for every item
- Syntax: for variable in list:



```python
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(magician)
```

Looping Through a List

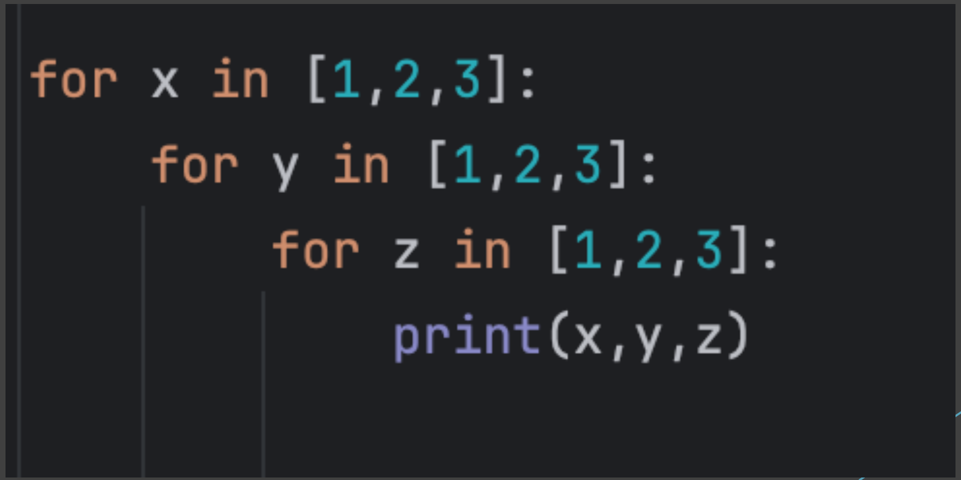
- Use a for loop to repeat actions for every item
- Syntax: for variable in list:
- Indentation is required



```python
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(magician)
```

By the Way: Indentation

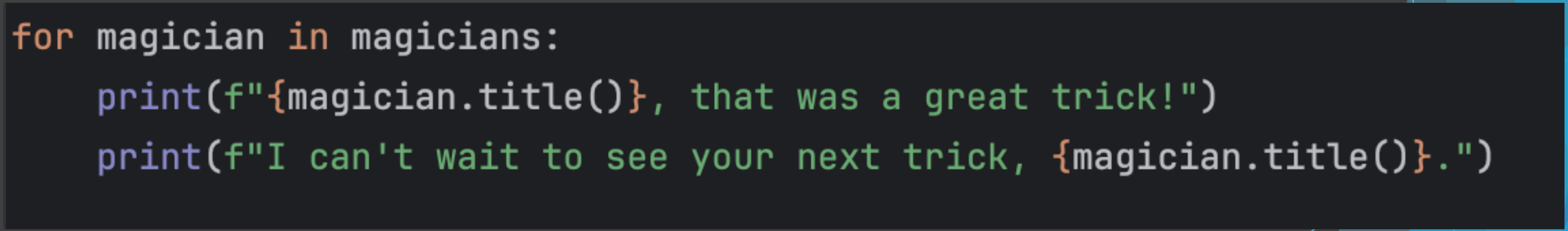
- Python uses indentation to group code blocks
- All indented lines belong to the loop
- Non-indented code runs after the loop ends
- This makes Python readable, but indentation must be exact



```python
for x in [1,2,3]:
    for y in [1,2,3]:
        for z in [1,2,3]:
            print(x,y,z)
```

Looping Through a List

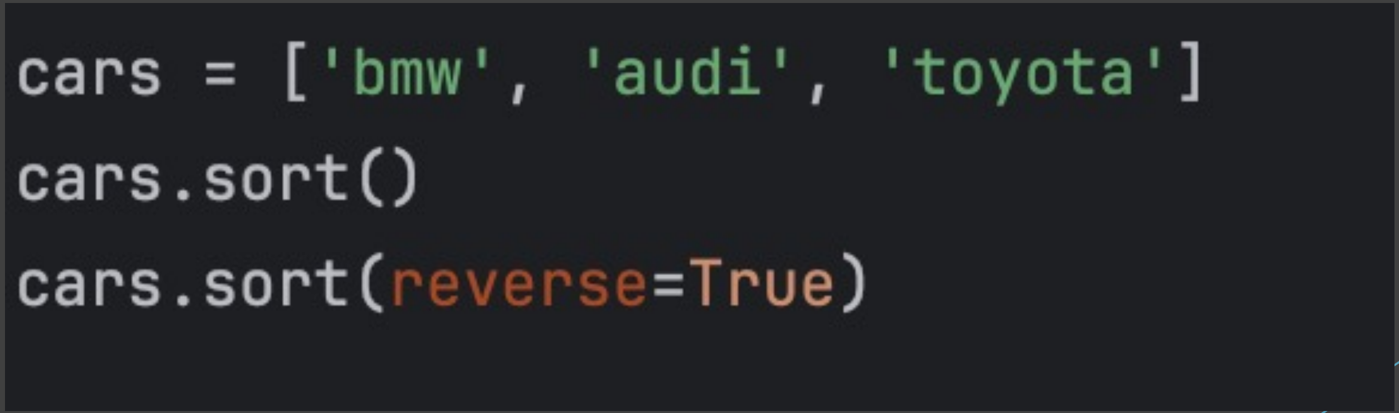
- You can do multiple things in a loop
- Each indented line runs for every item



```python
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.")
```

Sorting a List

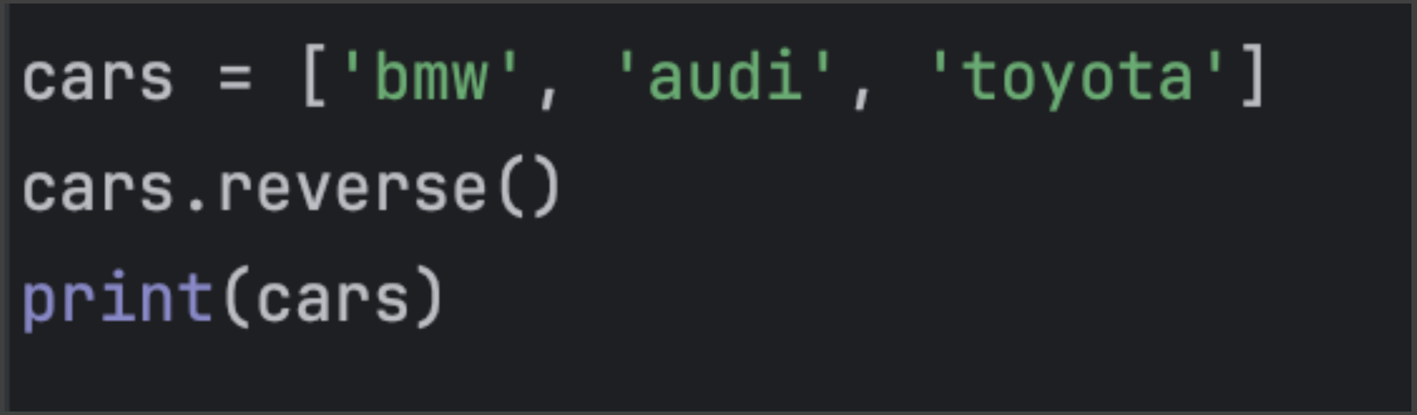
- sort() changes the list permanently
- sorted() returns a new sorted list
- Use reverse=True for reverse order



```
cars = ['bmw', 'audi', 'toyota']
cars.sort()
cars.sort(reverse=True)
```

Reversing a List

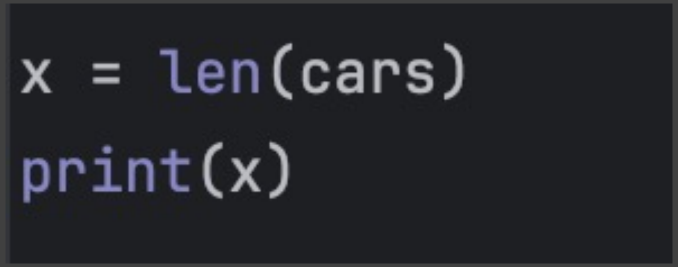
- Use reverse() to flip the list order
- Not the same as reverse sorting!



```
cars = ['bmw', 'audi', 'toyota']
cars.reverse()
print(cars)
```

Finding List Length

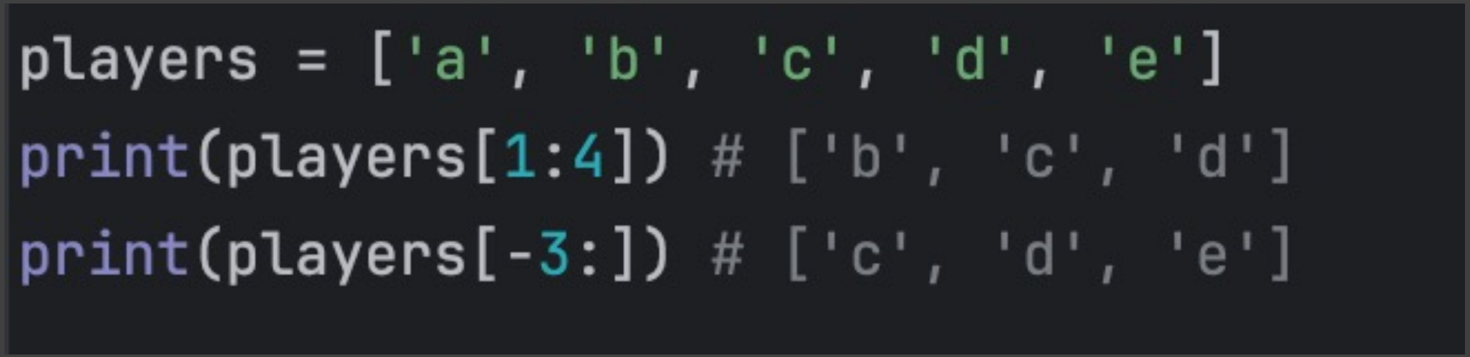
- Use len() to count how many items



```
x = len(cars)
print(x)
```

Slicing a List

- Get part of a list: list[start:end]
- End index is not included



```python
players = ['a', 'b', 'c', 'd', 'e']
print(players[1:4]) # ['b', 'c', 'd']
print(players[-3:]) # ['c', 'd', 'e']
```

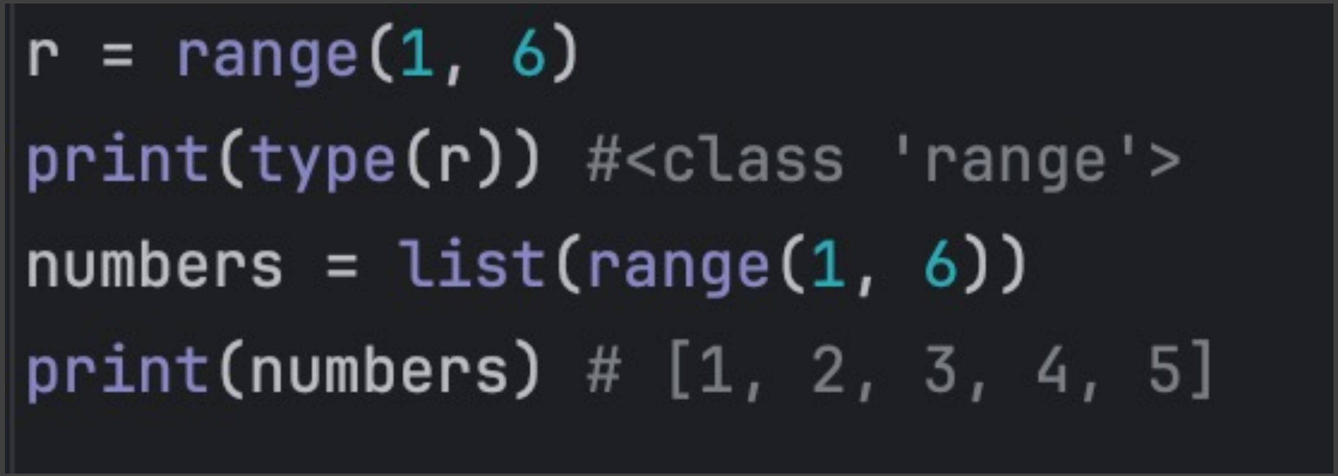
Key Takeaways

- for loops repeat actions for every item
- Indentation defines what is inside the loop
- Use sort, reverse, len, and slicing to organize your data

Numerical Lists, Comprehensions

Using range() to Generate Numbers

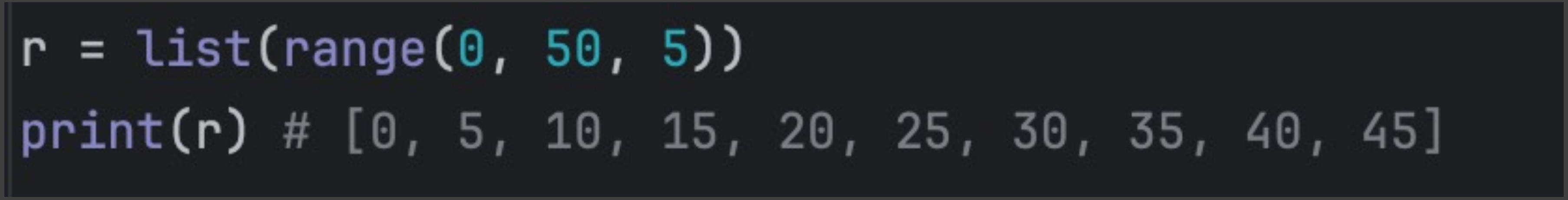
- range(start, stop) creates a sequence of numbers
- stop value is excluded
- Combine with list() to get a full list



```python
r = range(1, 6)
print(type(r)) #<class 'range'>
numbers = list(range(1, 6))
print(numbers) # [1, 2, 3, 4, 5]
```

range() with Steps

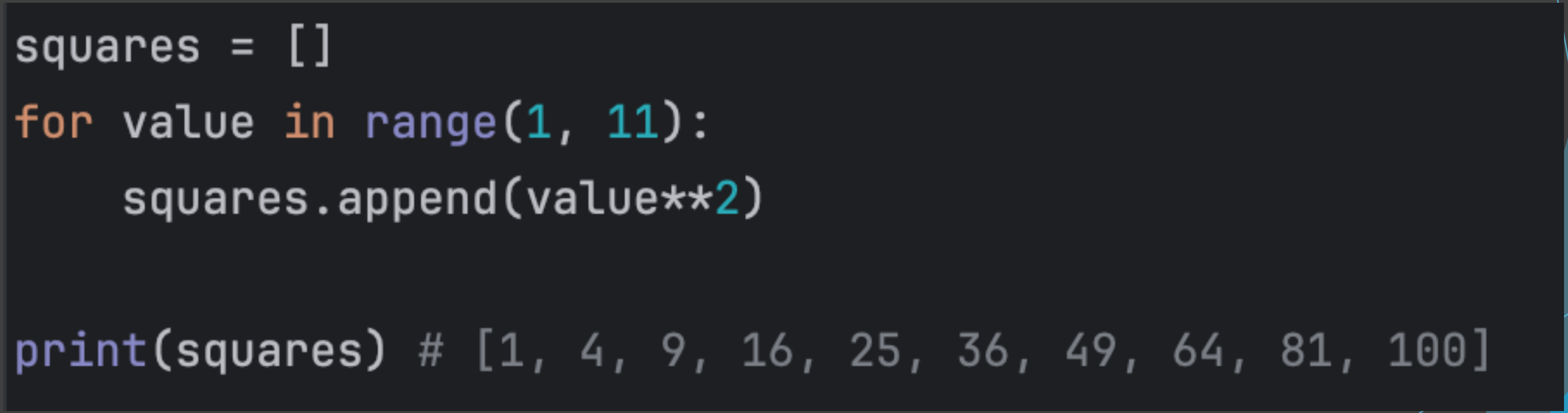
- Use a third argument to skip values



```python
r = list(range(0, 50, 5))
print(r) # [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

Squares with Loops

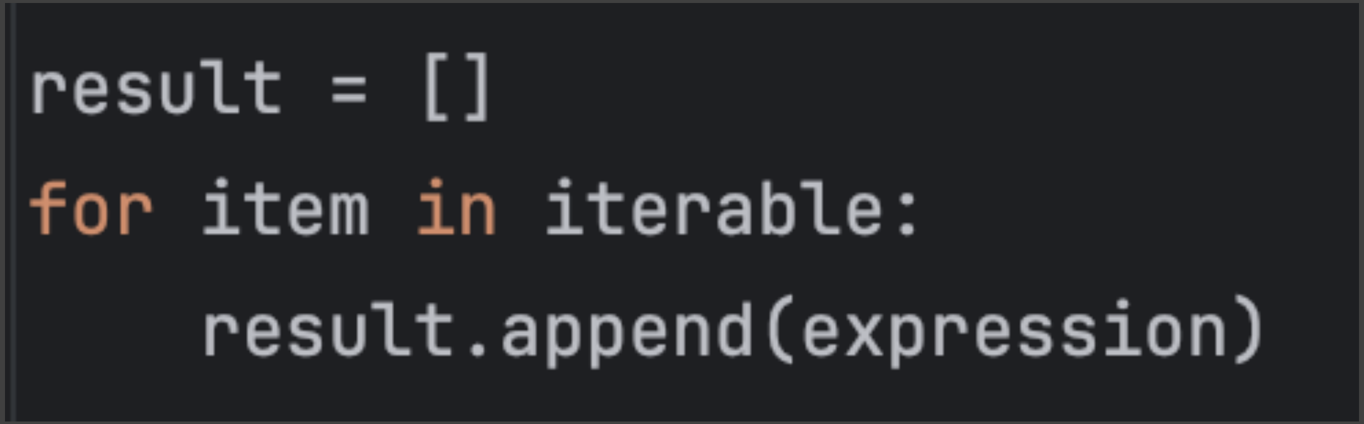
- Use a loop and append to build a list of squares



```
squares = []
for value in range(1, 11):
    squares.append(value**2)

print(squares) # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

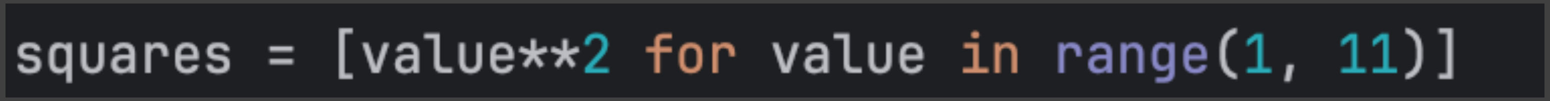
List Comprehensions

- A list comprehension is a one-line shortcut for creating a new list by looping through an iterable and applying an expression.
- Basic syntax: result = [expression for item in iterable]



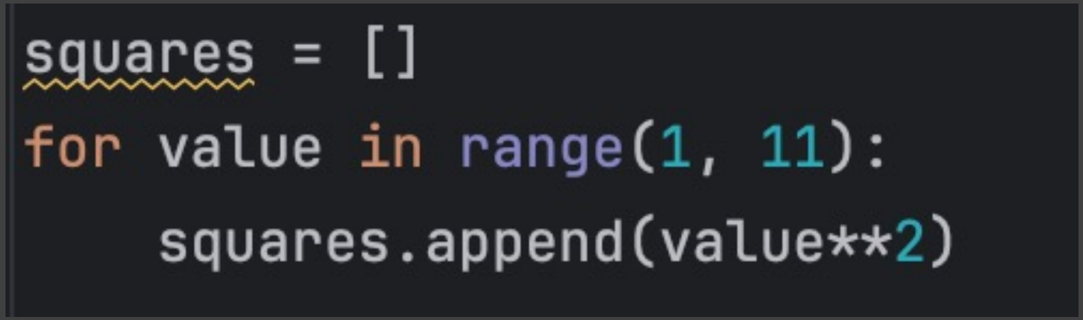
```
result = []
for item in iterable:
    result.append(expression)
```

List Comprehensions



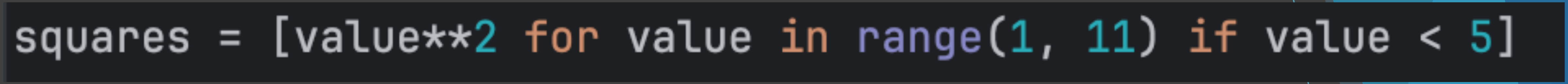
```python
squares = [value**2 for value in range(1, 11)]
```

- Loop through numbers from 1 to 10
- Square each number
- Store each result in the list squares



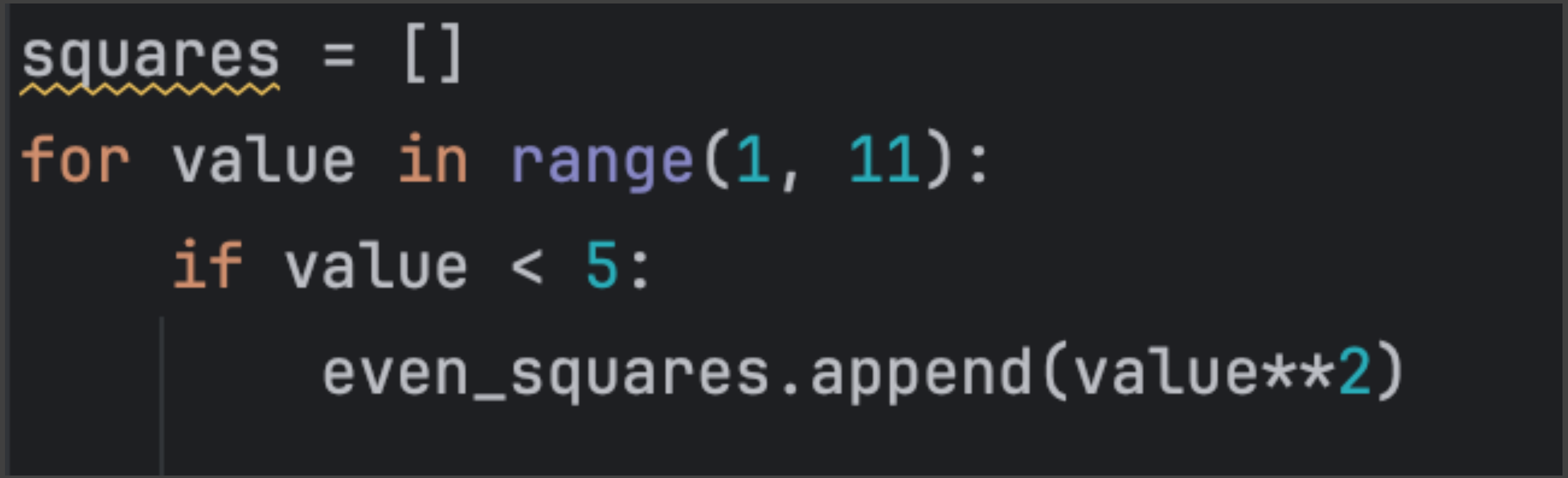
```python
squares = []
for value in range(1, 11):
    squares.append(value**2)
```

List Comprehensions



```python
squares = [value**2 for value in range(1, 11) if value < 5]
```

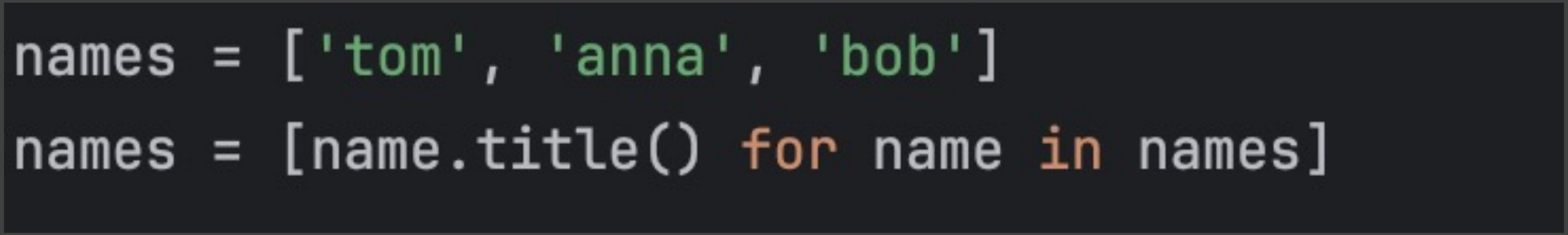
- Only square the even numbers from 1 to 10



```python
squares = []
for value in range(1, 11):
    if value < 5:
        even_squares.append(value**2)
```

List Comprehensions

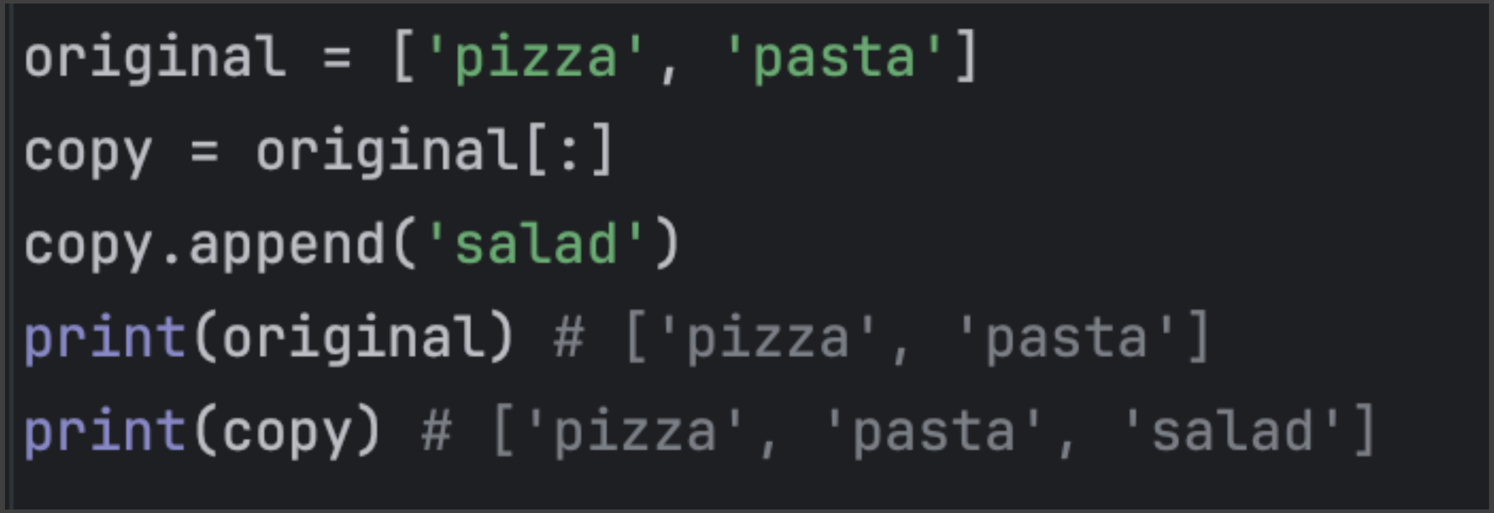
- Very flexible



```python
names = ['tom', 'anna', 'bob']
names = [name.title() for name in names]
```

Copying a list safely

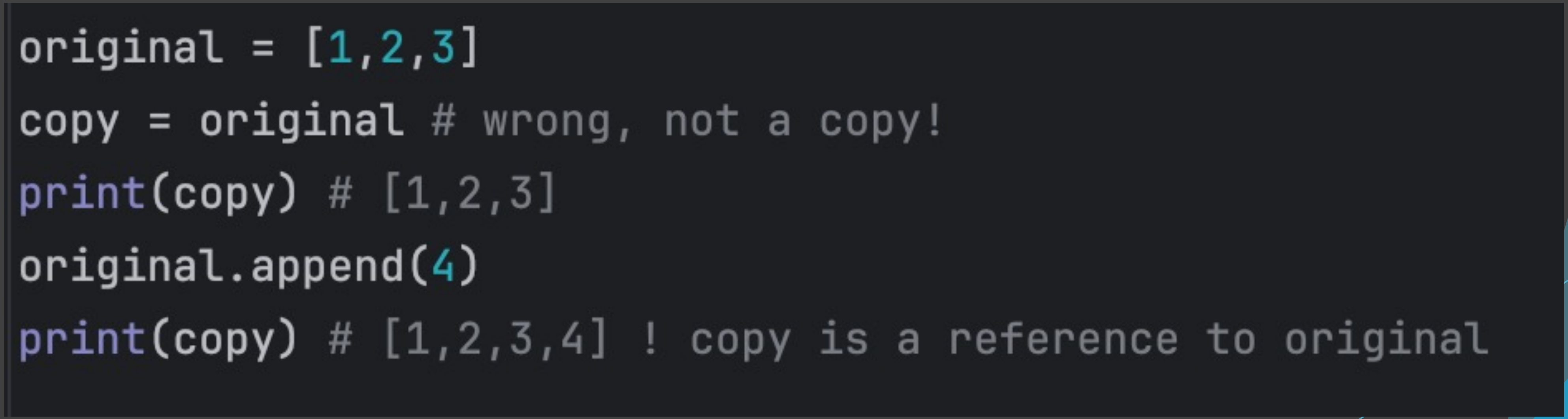
- Use slicing ([:]) to copy a list
- Avoid direct assignment, which creates a reference



```python
original = ['pizza', 'pasta']
copy = original[:]
copy.append('salad')
print(original) # ['pizza', 'pasta']
print(copy) # ['pizza', 'pasta', 'salad']
```

Copying a list safely

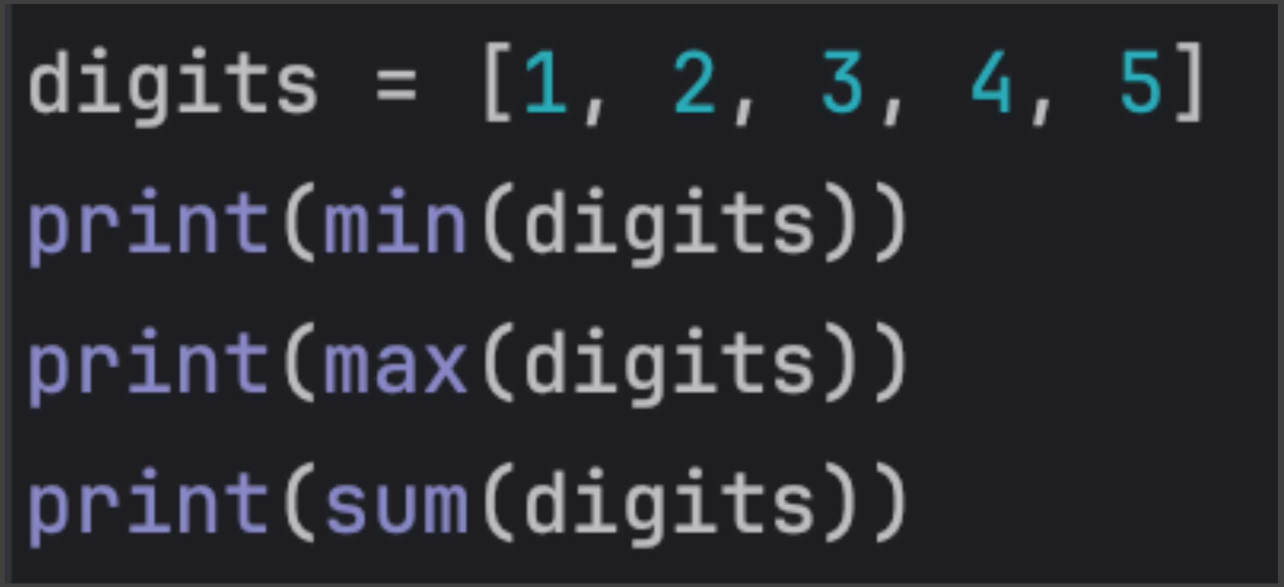
- Use slicing ([:]) to copy a list
- Avoid direct assignment, which creates a reference



```python
original = [1,2,3]
copy = original # wrong, not a copy!
print(copy) # [1,2,3]
original.append(4)
print(copy) # [1,2,3,4] ! copy is a reference to original
```

Simple List Math

- Use min(), max(), sum() on number lists



```python
digits = [1, 2, 3, 4, 5]
print(min(digits))
print(max(digits))
print(sum(digits))
```

Key Takeaways

- Use range() to create number sequences
- Use list comprehensions for compact logic
- Use min, max, sum for quick analysis

What does [x**2 for x in range(1, 4)] return?

0	0	0	0
[1, 4, 9]	[2, 4, 6]	[1, 2, 3, 4]	[1, 8, 27]

Which of these creates a true copy of a list?

0	0	0	0
copy = original[:]	copy = original	copy = list(original)	copy = original.copy()

Exercises

Exercise 1

- Create a list of at least 3 names and print each using indexing
- Print a personalized message for each name using f-strings
- Make a list of transport modes and print statements like "I would like to own a ___"

Exercise 2

- Make a guest list and send each person an invitation
- Replace one guest who can't come and resend invitations
- Insert guests at beginning, middle, and end, then print all invitations
- Shrink guest list to 2 using pop(), apologize, then delete all

Exercise 3

- List 3 pizzas, loop to print them all, then "I really love pizza!"
- List 3 animals and say why each makes a good pet; end with summary
- List 5 places. Print in original, sorted, reversed order (sorted() and reverse())

Exercise 4

- Use a loop to print numbers from 1 to 20
- List numbers to 1 million, use min(), max(), sum() on the list
- Confirm list starts at 1 and ends at 1 million
- Print odd numbers from 1 to 20
- Print multiples of 3 from 3 to 30
- List cubes of 1-10 using a loop
- List cubes of 1-10 using list comprehension
- Copy a list and show both are different

To join, go to: **ahaslides.com/GFO47**

AhaSlides

Which topic felt hardest today?

Slicing	Comprehensions	Looping
0	0	0

0 0/50

To join, go to: **ahaslides.com/GFO47**

AhaSlides

The speed of the session was too…

Fast

Strongly disagree

Strongly agree

1

5

Get Feedback

0 0/50