Python For Loop with Break

ALGORITHM
WITH CHELSEA

Review: List Traversal

Traverse means to visit each item, one by one.

Example:

```
numbers = [3, 1, 4]

for x in numbers:
    print(x)
```

- `x` becomes each item in the list, one by one.
- The code inside the loop must be **indented** (4 spaces).

What does `break` do?

`break` exits the loop immediately.

Example:

```
for x in [1, 2, 3, 4, 5]:
    if x == 3:
        break
    print(x)
```

What does `break` do?

`break` exits the loop immediately.

Example:

```python
for x in [1, 2, 3, 4, 5]:
    if x == 3:
        break
    print(x)
```

Output:

```
1
2
```

The loop stops when `x` becomes 3. So `print(x)` does not run for `x = 3`.

Search without `break`

We want to know if `target` is inside the list.

Example:

```
numbers = [4, 7, 2, 9, 5]
target = 9
```

Search without `break`

We want to know if `target` is inside the list.

Example:

```
numbers = [4, 7, 2, 9, 5]
target = 9
```

Idea:

- Use a Boolean variable `found`.
- Start with `found = False`.
- Traverse: visit every item of `numbers`.
- If we see the target, set `found = True`.

Search without `break`

Python code:

```python
numbers = [4, 7, 2, 9, 5]
target = 9

found = False

for x in numbers:
    if x == target:
        found = True

print(found)
```

Search without `break`

Python code:

```
numbers = [4, 7, 2, 9, 5]
target = 9

found = False

for x in numbers:
    if x == target:
        found = True

print(found)
```

Output:

```
True
```

Search without `break` (Another Example)

Python code:

```python
numbers = [4, 7, 2, 9, 5]
target = 4  # target is the first item

found = False

for x in numbers:
    if x == target:
        found = True

print(found)
```

- We find the target on the 1st iteration.
- The loop keeps checking the rest of the list even after we found the target.
- We can break as soon as we find the target.

Search with `break`

Python code:

```
numbers = [4, 7, 2, 9, 5]
target = 4

found = False

for x in numbers:
    if x == target:
        found = True
        break  # once we found the target, stop the loop

print(found)
```

Search with `break`

Python code:

```
numbers = [4, 7, 2, 9, 5]
target = 4

found = False

for x in numbers:
    if x == target:
        found = True
        break  # once we found the target, stop the loop

print(found)
```

Output:

```
True
```

Why is `break` helpful?

- Suppose `numbers` has 1 million items.
- If `target` is near the front, `break` saves a lot of work.
- You get the same answer, but faster.

Q1 🔎 Stop Early

What is the output?

```
numbers = [2, 7, 8, 5, 8, 9]

for x in numbers:
    if x == 8:
        break
    print(x)
```

Q1 🔎 Stop Early

What is the output?

```python
numbers = [2, 7, 8, 5, 8, 9]

for x in numbers:
    if x == 8:
        break
    print(x)
```

Output:

```
2
7
```

Search a Student Name

Goal:

- Search `target` in `students = ["Ava", "Ben", "Cody", "Dina"]`
- If `target` is in the list, print its **index**.
- If not found, print **-1**.

Search a Student Name

Goal:

- Search `target` in `students = ["Ava", "Ben", "Cody", "Dina"]`
- If `target` is in the list, print its **index**.
- If not found, print **-1**.

Idea:

- Start with `result = -1`.
- List size: `n = len(students)`.
- Loop over the index in `range(n)`.
- If we found `target`, assign its index to `result`, and then `break`.

Search a Student Name

Python code:

```python
students = ["Ava", "Ben", "Cody", "Dina"]
target = "Cody"

result = -1
n = len(students)   # 4

for i in range(n):  # 0, 1, 2, 3
    if students[i] == target:
        result = i
        break

print(result)
```

Search a Student Name

Python code:

```python
students = ["Ava", "Ben", "Cody", "Dina"]
target = "Cody"

result = -1
n = len(students)   # 4

for i in range(n):  # 0, 1, 2, 3
    if students[i] == target:
        result = i
        break

print(result)
```

Output:

```
2
```

Search a Student Name (Not Found)

Python code:

```python
students = ["Ava", "Ben", "Cody", "Dina"]
target = "Chelsea"

result = -1
n = len(students)   # 4

for i in range(n):  # 0, 1, 2, 3
    if students[i] == target:
        result = i
        break

print(result)
```

Search a Student Name (Not Found)

Python code:

```python
students = ["Ava", "Ben", "Cody", "Dina"]
target = "Chelsea"

result = -1
n = len(students)   # 4

for i in range(n):  # 0, 1, 2, 3
    if students[i] == target:
        result = i
        break

print(result)
```

Output:

```
-1
```

Q2 🙋🏻‍♀️ Print Student Names

What is the output?

```
students = ["Ava", "Ben", "Cody", "Dina"]
target = "Cody"

n = len(students)    # 4

for i in range(n):  # 0, 1, 2, 3
    print(students[i])
    if students[i] == target:
        break
```

Q2 🙋‍♀️ Print Student Names

What is the output?

```
students = ["Ava", "Ben", "Cody", "Dina"]
target = "Cody"

n = len(students)   # 4

for i in range(n):  # 0, 1, 2, 3
    print(students[i])
    if students[i] == target:
        break
```

Output:

```
Ava
Ben
Cody
```

List Index `-1` and `-2`

Python allows **negative indices**:

- `-1` means the **last** item.
- `-2` means the **second-to-last** item.

Example:

```
nums = [10, 20, 30, 40]

print(nums[-1])  # 40
print(nums[-2])  # 30
```

Fibonacci Numbers

The **Fibonacci numbers** start like this:

```
1, 1, 2, 3, 5, 8, 13, 21, ...
```

Each number is the sum of the previous two.

Fibonacci Numbers up to 100

Goal: Build a list of Fibonacci numbers that are `<= 100`.

Idea:

- Start with list `fib = [1, 1]`.
- The next number is `nxt = fib[-2] + fib[-1]`.
- If `nxt > 100`, we stop the loop.
- We don't know how many Fibonacci numbers up to 100, so we use a big range.

Fibonacci Numbers up to 100

Python code:

```python
fib = [1, 1]

for _ in range(1000):    # 1000 is big enough
    nxt = fib[-2] + fib[-1]
    if nxt > 100:
        break
    fib.append(nxt)

print(fib)
```

Remark: `_` means we don’t use this variable.

Fibonacci Numbers up to 100

Python code:

```python
fib = [1, 1]

for _ in range(1000):    # 1000 is big enough
    nxt = fib[-2] + fib[-1]
    if nxt > 100:
        break
    fib.append(nxt)

print(fib)
```

Output:

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Q3 🖨️ Fibonacci Numbers

Change: Moved `append()` before `if`.

```
fib = [1, 1]

for _ in range(1000):
    nxt = fib[-2] + fib[-1]
    fib.append(nxt)  # moved here
    if nxt > 100:
        break
    # previously, append was here

print(fib)
```

What is the output?

Q3 🖨️ Fibonacci Numbers

Change: Moved `append()` before `if`.

```
fib = [1, 1]

for _ in range(1000):
    nxt = fib[-2] + fib[-1]
    fib.append(nxt)  # moved here
    if nxt > 100:
        break
    # previously, append was here

print(fib)
```

Output:

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

Summary

- `break` stops the loop immediately.
- Use `break` to stop early so the program runs faster.
- Search: If the target is found, stop the loop by `break`.
- Negative indices:
  - `items[-1]` is the last item
  - `items[-2]` is the second-to-last item
- Fibonacci:
  - `nxt = fib[-2] + fib[-1]`
  - `fib.append(nxt)`

ALGORITHM
WITH CHELSEA