ITERATION IN PYTHON

Iteration: Loops

- A loop is a sequence of instructions to be repeated
- Definite and Indefinite
  - Definite: repeat exactly X times
  - Indefinite: repeat until some condition changes

for Loops

- A for statement is one way to create a loop in Python
  - Allows us to *repeat* statements a specific number of times
- Example

```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
```

will output

```
Warning
1
Warning
2
Warning
3
```

- The repeated statement(s) are known as the body of the loop
  - Must be indented the same amount in Python

for Loops

- General syntax

```
for <variable> in <sequence>:
    <body of the loop>
```

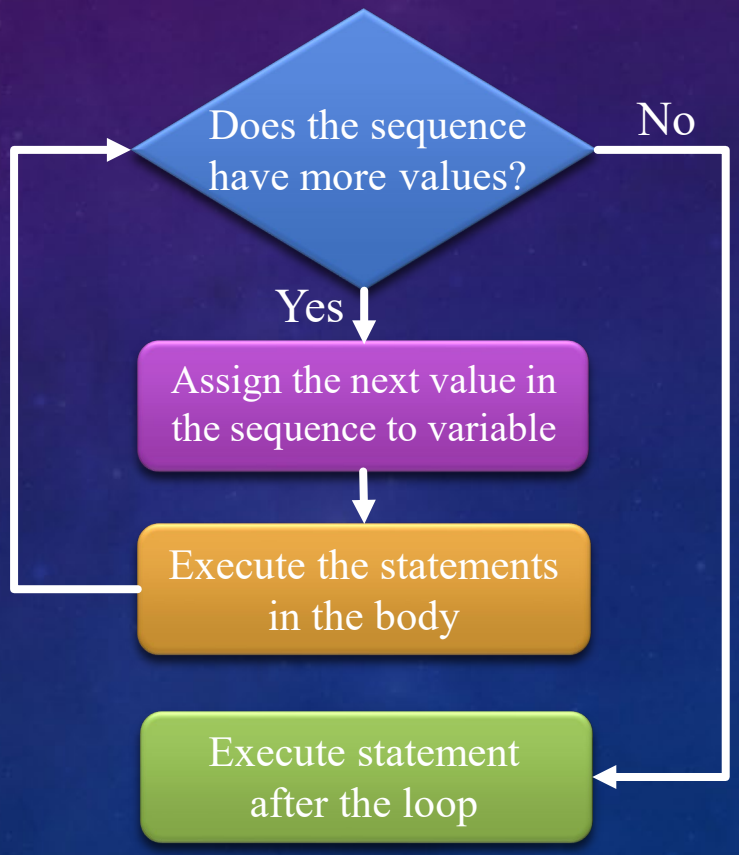
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
```

- In this case, our sequence is a sequence of values, but it could be any sequence (i.e. for word in list_of_words)
- For each value in the sequence:
  - The value is assigned to the variable
  - All statements in the body of the loop are executed using that value
- Once all values in the sequence have been processed, the program continues with the first statement after the loop

Executing a *for* Loop

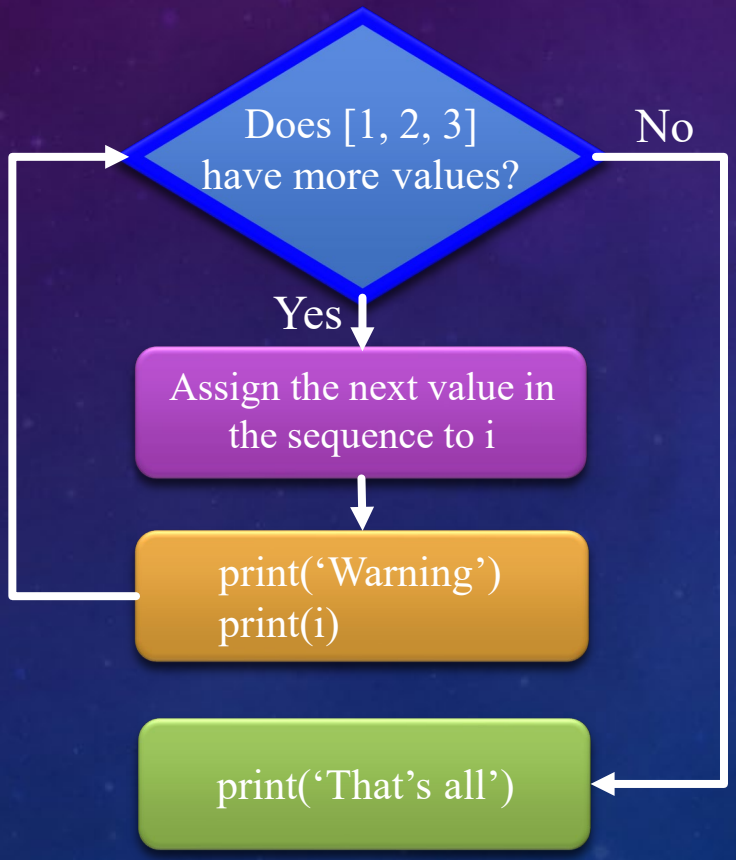
```
for <variable> in <sequence>:
    <body of the loop>
```

```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
```



Executing the Earlier Example

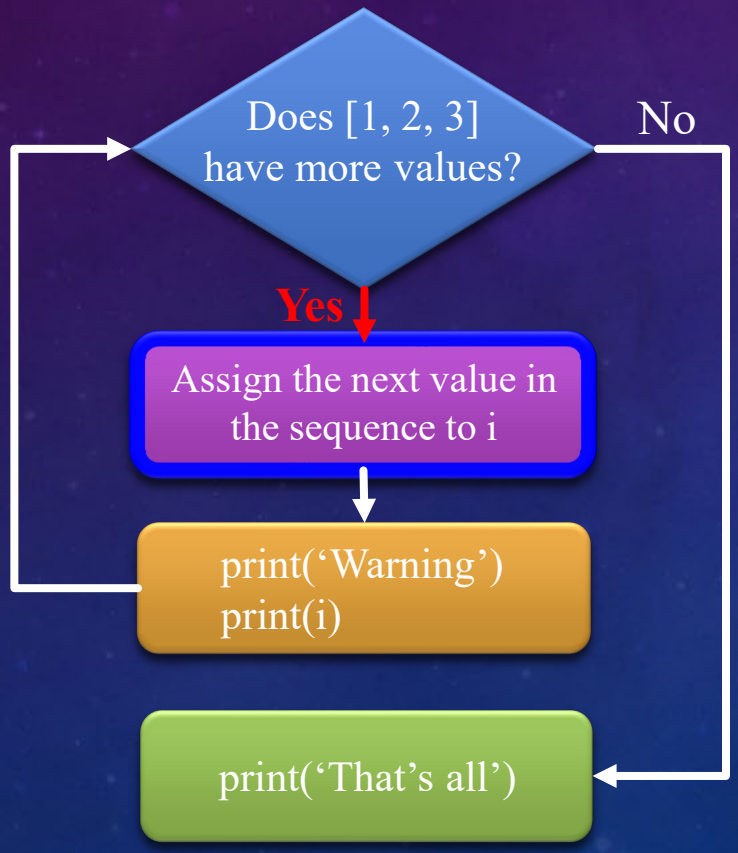
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes		

Executing the Earlier Example

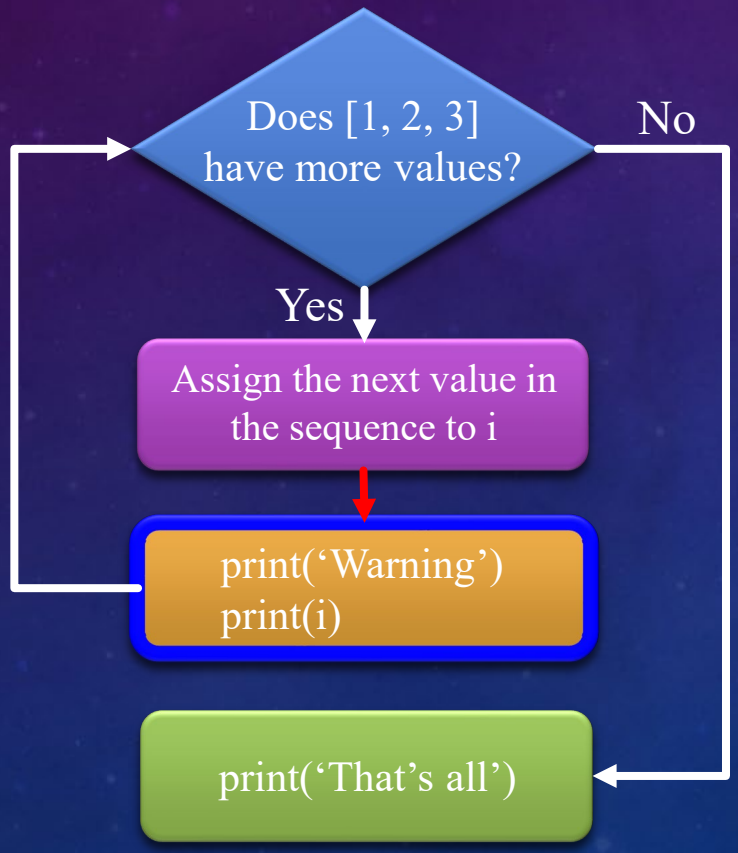
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes	1	

Executing the Earlier Example

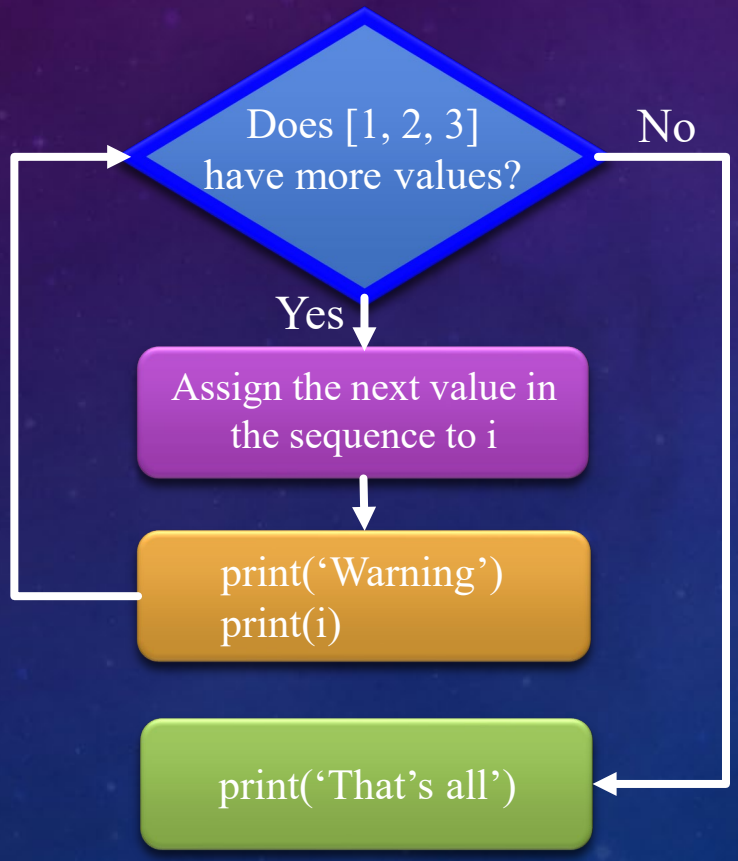
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes	1	Warning 1

Executing the Earlier Example

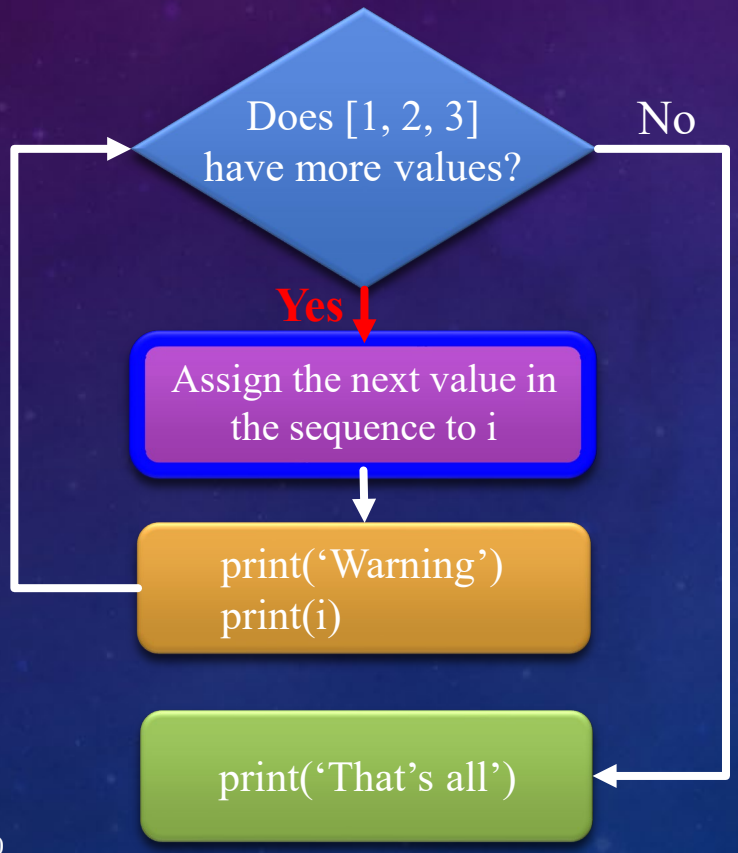
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes	1	`Warning` `1`
Yes		

Executing the Earlier Example

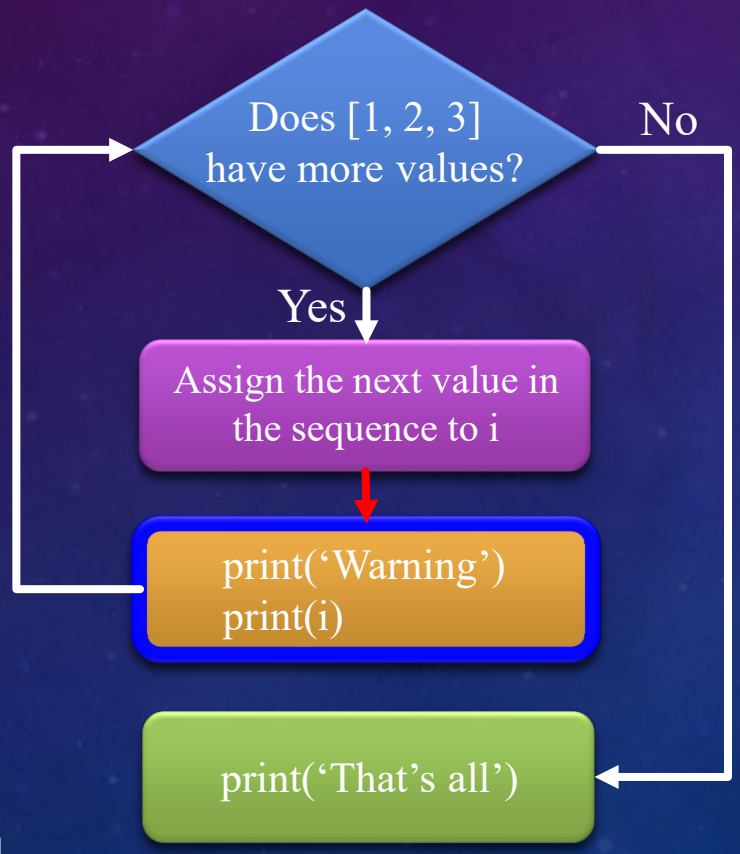
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes	1	`Warning` `1`
Yes	2	

Executing the Earlier Example

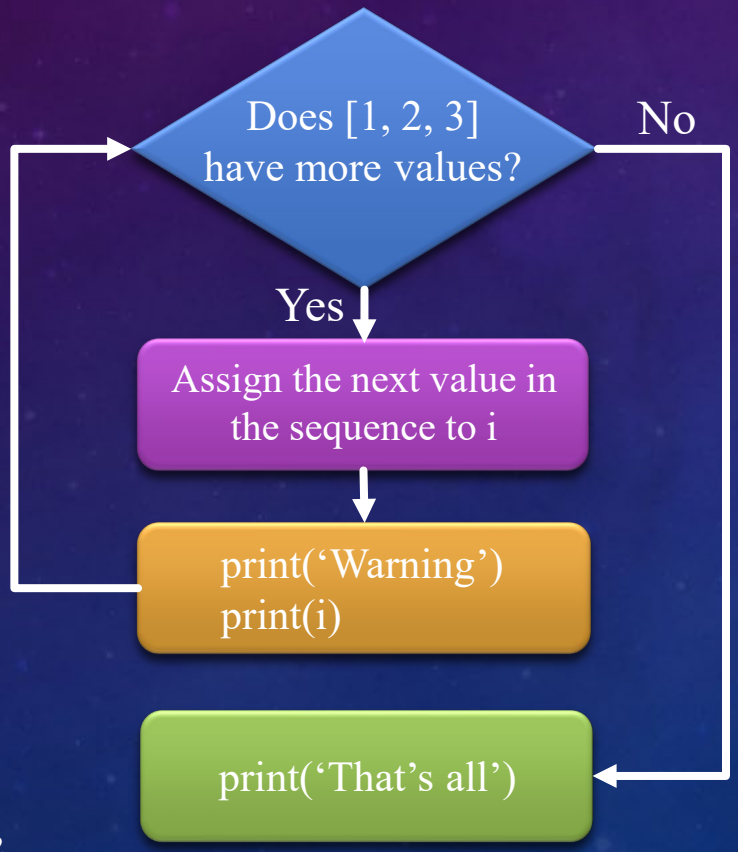
```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes	1	Warning 1
Yes	2	Warning 2

Executing the Earlier Example

```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```

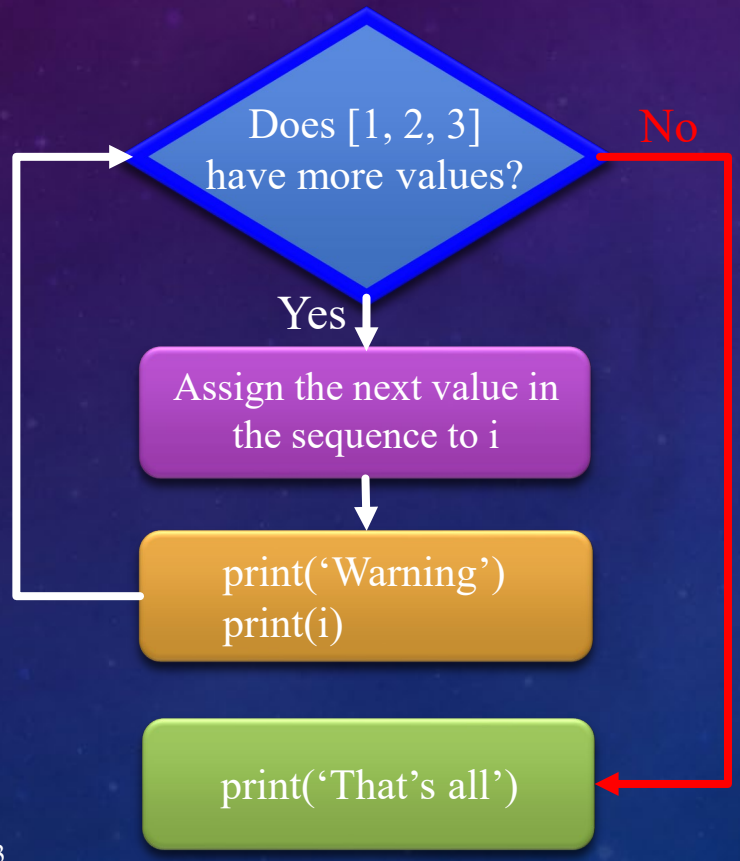


More?	i	Output / Action
Yes	1	`Warning` `1`
Yes	2	`Warning` `2`

** Skipping to the end of loop **

Executing the Earlier Example

```
for i in [1, 2, 3]:
    print('Warning')
    print(i)
print('That's all')
```



More?	i	Output / Action
Yes	1	`Warning 1`
Yes	2	`Warning 2`
Yes	3	`Warning 3`
No		

Simple Repetition Loops

- To repeat a loop's body N times

```
for i in range(N):
    <body of the loop>
```

← [0, 1, 2, …, N - 1]

- Example: What would this loop do?

```
for i in range(3):
    print('I'm feeling loopy!')
```

Simple Repetition Loops

- To repeat a loop's body N times

```
for i in range(N):    ← [0, 1, 2, …, N - 1]
    <body of the loop>
```

- Example

```
for i in range(3): ← [0, 1, 2]
    print('I'm feeling loopy!')
```

- Outputs

```
I'm feeling loopy!
I'm feeling loopy!
I'm feeling loopy!
```

Simple Repetition Loops

- To repeat a loop's body N times

```
for i in range(N):    ← [0, 1, 2, …, N - 1]
    <body of the loop>
```

- Example

```
for i in range(5): ← [0, 1, 2, 3, 4]
    print('I'm feeling loopy!')
```

- Outputs

```
I'm feeling loopy!
I'm feeling loopy!
I'm feeling loopy!
I'm feeling loopy!
I'm feeling loopy!
```

Simple Repetition Loops

- Another example

```
for i in range(7):
    print(i * 5)
```

← [0, 1, 2, 3, 4, 5, 6]

- How many repetitions? 7
- Output?

```
0
5
10
15
20
25
30
```

for Loops Are Definite Loops

- A definite loop is a loop in which the number of repetitions is fixed before the loop even begins
- In a for loop, # of repetitions = len(sequence)

```
for <variable> in <sequence>:
    <body of the loop>
```

How to Print the Warning 20 times?

```
for i in __________:
    print('Warning')
```

A. range(20)
B. [1] * 20
C. 'abcdefghijklmnopqrst'

These are all sequences with a length of 20!

D. Either A or B would work, but not C
E. A, B, or C would work

How to Print the Warning 20 times?

```
for i in __________:
    print('Warning')
```

A. range(20)
B. [1] * 20
C. 'abcdefghijklmnopqrst'

These are all sequences with a length of 20!

D. Either A or B would work, but not C
E. A, B, or C would work

Python Arithmetic Shortcuts

- Here are some *augmented assignment* statements that can be used in for loops
- Consider this code

```
age = 18
age = age + 1
```

- Instead of writing

```
age = age + 1
```

we can just write

```
age += 1
```

Python Arithmetic Shortcuts

Shortcut	Equivalent to
`var += expr`	`var = var + expr`
`var -= expr`	`var = var - expr`
`var *= expr`	`var = var * expr`
`var /= expr`	`var = var / expr`
`var //= expr`	`var = var // expr`
`var %= expr`	`var = var % expr`
`var **= expr`	`var = var ** expr`

where *var* is a variable, *expr* is an expression

Important: the = must come *after* the other operator

`+=` is correct

`=+` is not!

How to Add the Numbers in the list *vals*?

```
vals = [10, 20, 30, 40, 50]
result = 0
for _______________:
    result += _______________
print(result)
```

First blank	Second blank
A. x in vals	x
B. x in vals	vals[x]
C. i in range(len(vals))	vals[i]

D. Either A or B would work, but not C

E. Either A or C would work, but not B

How to Add the Numbers in the list *vals*?

```
vals = [10, 20, 30, 40, 50]
result = 0
for _______________:
    result += _______________
print(result)
```

First blank	Second blank
A. x in vals	x
B. x in vals	vals[x]
C. i in range(len(vals))	vals[i]

D. Either A or B would work, but not C

E. Either A or C would work, but not B

Using a Loop to Sum a List of Numbers

```
vals = [10, 20, 30, 40, 50]
result = 0  ← Accumulator variable
for x in vals:
    result += x  ← Gradually accumulates the sum
print(result)
```

x	result
	0
10	10
20	30
30	60
40	100
50	150

Another Example

- What would this code output?

```
num_iters = 0
for val in [2, 4, 16, 8, 10]:
    num_iters += 1
    print(val * 10)
print(num_iters)
```

- Use a table to help you

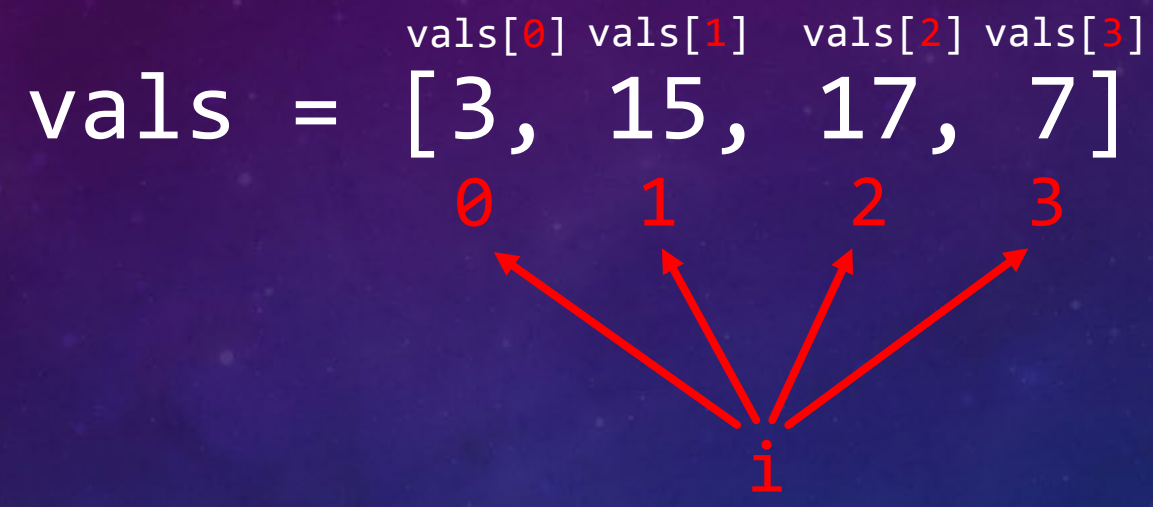
More?	val	num_iters	Output
Yes	2	1	20
Yes	4	2	40
Yes	16	3	160
Yes	8	4	80
Yes	10	5	100
No		5	

Element-based *for* Loop

```
vals = [3, 15, 17, 7]
```


```
result = 0
for x in vals:
    result += x
print(result)
```

Index-based *for* Loop




```
result = 0
for i in range(len(vals)):
    result += vals[i]
print(result)
```

Tracing an Index-based Cumulative Sum

```
vals = [10, 20, 30, 40, 50]
result = 0
for i in range(len(vals)):
    result += vals[i]
print(vals)
```

i	vals[i]	result
		0
0	10	10
1	20	30
2	30	60
3	40	100
4	50	150

What Is the Output of This Program?

```
vals = [5, 7, 7, 2, 3, 3, 5]
result = 0
for i in range(len(vals)): → range(7) → [0,1,2,3,4,5,6]
    if vals[i] == vals[i - 1]:
        result += 1
print(result)
```

i	vals[i]	vals[i - 1]	result
			0
0	5	5	1
1	7	5	1
2	7	7	2
3	2	7	2
4	6	2	2
5	6	6	3
6	5	6	3

A. 0
B. 1
C. 2
D. 3
E. 7

What Is the Output of This Program?

```
vals = [5, 7, 7, 2, 3, 3, 5]
result = 0
for i in range(len(vals)): → range(7) → [0,1,2,3,4,5,6]
    if vals[i] == vals[i - 1]:
        result += 1
print(result)
```

i	vals[i]	vals[i - 1]	result
			0
0	5	5	1
1	7	5	1
2	7	7	2
3	2	7	2
4	3	2	2
5	3	3	3
6	5	3	3

A. 0
B. 1
C. 2
D. 3
E. 7

More on Cumulative Arithmetic

- Here is a loop-based factorial in Python

```
result = 1
for x in range(n): ← [0, 1, 2, …, n - 1]
    result *= x ← 1 = 1 * 0 ...
print(result) ← result = 0
```

- Does this snippet work? No!

More on Cumulative Arithmetic

- How can we make this do what we want?

```
result = 1
for x in range(__________):
    result *= x
print(result)
```

← Fill the blank

```
range([start], stop, [step])

  start: Starting number of the sequence
  stop: Generate numbers up to but not including this number
  step: Difference between each number in the sequence
```

More on Cumulative Arithmetic

- How can we make this do what we want?

```
result = 1
for x in range(1, n + 1):
    result *= x
print(result)
```

```
range([start], stop, [step])

  start: Starting number of the sequence
  stop: Generate numbers up to but not including this number
  step: Difference between each number in the sequence
```

More on Cumulative Arithmetic

- Here's a loop-based factorial in Python

```
result = 1 ← Accumulator variable
for x in range(1, n + 1):
    result *= x ← Accumulates the factorial
print(result)
```

- Is this loop element-based or index-based?

  Element-based. The loop variable takes on elements from the sequence that we're processing

Cumulative Arithmetic with Strings

- Write a snippet to remove all the vowels of a string

```
s = ‘engineering’
***
Try to fill
                the missing code
***
print(result)
ngnrng
```

- If `s = ‘industrial’`, we get `‘ndstrl’`
- How do we do this?

Cumulative Arithmetic with Strings

```
s = ‘engineering’
result = ‘’ ← Accumulator variable
for char in s:
    if char not in ‘aeiou’:
        result += char ← Accumulates the result
print(result)
```

Char	Result
	‘’
‘e’	‘’ (no change)
‘n’	‘’ + ‘n’ → ‘n’
‘g’	‘n’ + ‘g’ → ‘ng’
‘i’	‘ng’ (no change)
‘n’	‘ng’ + ‘n’ → ‘ngn’
...	...

List Comprehension

- List comprehensions use *for* loops within brackets to construct a list
- We can create a list of integers up to *i* by using list comprehensions

```
result = [i for i in range(5)]
print(result)
[0, 1, 2, 3, 4]
result = [x**2 for x in range(7)]
print(result)
[0, 1, 4, 9, 16, 25, 36]
```

- Syntax: [*expression for item in sequence*]
- The above syntax is useful for creating lists in one line. It includes all items in that list
- You can also use list comprehensions to modify an existing list

List Comprehension

- We can include if-else statements to perform more complex operations
- Let's try to remove vowels of a string with list comprehensions

```
s = 'engineering'
result = [char for char in s if char not in 'aeiou']
print(result)
```

```
ngnrng
```

- This syntax allows us to use complex expressions to make a list in a single line
- Two valid formats
  - [*expression1 if condition else expression2 for item in sequence*]
  - [*expression for item in sequence if condition*]

What Is the Output of the Following Expression?

```
nums1 = [i for i in range(10)]
nums2 = [2 * i if i % 2 == 0 else i for i in nums1]
print(nums2)
```

A. [0, 1, 4, 3, 8, 5, 12, 7, 16, 9]
B. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
C. [0, 1, 4, 3, 4, 5, 12, 7, 16, 9, 20]
D. [0, 4, 8, 12, 16]
E. Error message

What Is the Output of the Following Expression?

```
nums1 = [i for i in range(10)]
nums2 = [2 * i if i % 2 == 0 else i for i in nums1]
print(nums2)
```

A. [0, 1, 4, 3, 8, 5, 12, 7, 16, 9]
B. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
C. [0, 1, 4, 3, 4, 5, 12, 7, 16, 9, 20]
D. [0, 4, 8, 12, 16]
E. Error message

What Does This Program Output?

```
s = ‘time to think! ’
result = ‘’
for i in range(len(s)):
    if s[i - 1] == ‘ ’:
        result += s[i]
print(result)
```

A. tt
B. ttt
C. tothink!
D. timetothink!
E. None of the above

What Does This Program Output?

```
s = ‘time to think! ’
result = ‘’
for i in range(len(s)):
    if s[i - 1] == ‘ ’:
        result += s[i]
print(result)
```

Can you do the same thing using an element-based for loop? Why or why not?

A. tt
B. ttt
C. tothink!
D. timetothink!
E. None of the above