At the end of this lecture, students should:
• understand the Python range() function and be able to use it to define a series of values
• understand the for...in loop structure used with the range() function
• be able to define a for...in range() loop to implement counter-controlled repetition
• be able to convert a for...in range() loop into an equivalent while loop and vice versa

The Python range() function
The Python range() function defines a sequence of integer values within boundaries.
The range() function has the following syntax: \texttt{range(start, stop, step)}
where the three arguments are:
\texttt{start} - the lower bound (included) of the sequence defined,
\texttt{stop} - the upper bound (excluded) of the sequence defined,
\texttt{step} - the increment between each number in the sequence defined.

Some examples:
• range(1, 10, 2) defines the sequence 1, 3, 5, 7, 9
• range(5, 20, 6) defines the sequence 5, 11, 17
• range(14, 4, -3) defines the sequence 14, 11, 8, 5
• range(0, 7, 1) defines the sequence 0, 1, 2, 3, 4, 5, 6
The Python `range()` function continued

\[
\text{range(start, stop, step)}
\]

If the step is omitted, the default step is 1.
- `range(0, 7)` defines the sequence 0, 1, 2, 3, 4, 5, 6

If both the start and the step are omitted, the sequence starts from 0 with a step increment of 1.
- `range(5)` defines the sequence 0, 1, 2, 3, 4,
- `range(7)` defines the sequence 0, 1, 2, 3, 4, 5, 6

**Note that printing a range object does NOT print the defined sequence of integers, i.e.,**
- `print(range(6))` does NOT print the numbers 0 1 2 3 4 5

The step cannot be 0:
- `range(0, 7, 0)` gives an error

If the step is negative then the start value should be greater than the stop value.
- `range(14, 4, -3)` defines the sequence 14, 11, 8, 5
- `range(4, 14, -3)` defines an **empty range of numbers**

If the step is positive then the start value should be smaller than the stop value.
- `range(14, 4, 3)` defines an **empty range of numbers**
- `range(4, 14, 3)` defines the sequence 4, 7, 10, 13

**Iteration – for...in loops**

The following `while` loop executes exactly 100 times (for `count = 0` to `count = 99`). The variable, `count`, controls the number of times the loop body is executed.
```
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1
print("Done!")
```

The `for in range(...)` loop provides a compact structure for counter-controlled loops.
```
for count in range(0, 100):
    print("Programming is fun!")
print("Done!")
```

```
for value in range(0, 100):
    print("Programming is fun!")
```

Note that in the `for...in loop` on the previous slide the name used for the loop variable can be **any identifier**. The following two sections of code behave in exactly the same way.
```
for number in range(3, 7):
    print(number * 5)
```

```
for number in range(0, 100):
    print("Programming is fun!")
```

Note that in the `for...in loops` above, the loop body is executed for each value in the series of numbers defined by the `range()` function. In the body of the loop, the loop variable takes on each value of the series of numbers defined by the `range()` function, e.g.,
```
for value in range(0, 5):
    print(value)
```

```
for number in range(3, 7):
    print(number * 5)
```

```
for value in range(0, 5):
    print(value)
```

```
for number in range(3, 7):
    print(number * 5)
```

```
for value in range(0, 5):
    print(value)
```

```
for number in range(3, 7):
    print(number * 5)
```

```
for value in range(0, 5):
    print(value)
```

```
for number in range(3, 7):
    print(number * 5)
```
Give the output

def print_output():
    total = 0
    for number in range(9, 20):
        if number % 2 == 0 or number % 3 == 0:
            total = total + 1
    print(total)
def main():
    print_output()
main()

Complete the loops

Complete the for...in range() loop so that the output is:

for number in range(9, 20):
    if number % 2 == 0 or number % 3 == 0:
        total = total + 1
    print(total)

Complete the function

An amount doubles each year. Using a for...in range() loop complete the double_each_year() function which prints the growth of the parameter, (start_amt) for the given number of years (num_years). The first line printed by the function is the starting amount.
Each line of the output is numbered starting from the number 1. The function returns the final amount.

def double_each_year(start_amt, num_years):
    for i in range(num_years):
        print_series(starting with: start_amt
Starting with: 24
1: 48
2: 96
3: 192
4: 384
After 4 years: 384
Starting with: 235
1: 470
2: 940
3: 1880
After 3 years: 1880
Starting with: 15
1: 30
2: 60
3: 120
4: 240
5: 480
After 5 years: 480

Complete the function

Using a for...in range() loop complete the print_series() function which prints a series of numbers starting from the parameter value, start_num. The second number printed is the first number plus 1, the third number is the second number plus 2, the fourth number is the third number plus 3, and so on, e.g., a series of 8 numbers starting from the number 2 is:

for number in range(start_num, how_many):
    print(value, end = " ")

Complete the function

Starting with: 24
1: 48
2: 96
3: 192
4: 384
After 4 years: 384
Starting with: 235
1: 470
2: 940
3: 1880
After 3 years: 1880
Starting with: 15
1: 30
2: 60
3: 120
4: 240
5: 480
After 5 years: 480

def double_each_year(start_amt, num_years):
    double_each_year(24, 4)
    print("After 4 years:", double_each_year(24, 4))
    print("

print_series(2, 8)
print_series(5, 12)
print_series(16, 9)

main()
while loop vs for...in loops

Counter-controlled while loops can be converted into for...in range() loops and vice versa.

```python
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1
```

```python
for count in range(0, 100):
    print("Programming is fun!")
```

Not all while loops can be expressed using a for...in range(...) loop (only the ones for which we know exactly how many times the loop body is to be executed).

All for...in range() loops can be expressed as while loops.

```python
counter = 12
while counter < 260:
    count = count + 1
    total = total + top + bottom
    bottom = bottom + 2
    print("count: ", count, "sum: ", total)
```

```python
for bottom in range(0, top + 1, 2):
    count = count + 1
    total = total + top + bottom
print("count: ", count, "sum: ", total)
```

Same output?

Do the following two loops give the same output? If not, what is the difference in output and what change needs to be made in order to make the outputs of the two loops the same?

```python
top = 6
bottom = 0
count = 0
total = 0
while bottom < top:
    count = count + 1
    total = total + top + bottom
    bottom = bottom + 2
print("count: ", count, "sum: ", total)
```

```python
top = 6
count = 0
total = 0
for bottom in range(0, top + 1, 2):
    count = count + 1
    total = total + top + bottom
print("count: ",count,"sum: ",total)
```

Convert - while loop ←→ for...in loop

Convert the following while loop into a for...in range() loop:

```python
counter = 12
while counter < 260:
    print(counter)
    counter = counter + 10
```

Convert the following for...in range() loop into a while loop:

```python
for num in range(45, 3, -5):
    print(num * 2)
```

Complete the function

A perfect number is an integer that is equal to the sum of its divisors (including 1, excluding the number itself), e.g., 28 = 1 + 2 + 4 + 7 + 14. Complete the get_sum_of_divisors() function using a for...in range() loop for the iteration.

```python
28 is a perfect number
54 is NOT a perfect number
496 is a perfect number
```

```python
def get_sum_of_divisors(number):
    def check_perfection(number):
        if number == get_sum_of_divisors(number):
            print(number, "is a perfect number")
        else:
            print(number, "is NOT a perfect number")
    def main():
        check_perfection(28)
        check_perfection(54)
        check_perfection(496)
    main()
```
The `get_series_sum()` function returns the sum of the desired number of terms of the series:

\[ \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots. \]

e.g., `get_series_sum(4)` returns the sum of one half plus one quarter plus one eighth plus one sixteenth. Complete the function

```python
def get_series_sum(num_terms):
    # Your implementation here

def main():
    for num in range(1, 10):
        comment = "Terms: " + str(num) + ", sum:"
        print(comment, get_series_sum(num))
main()
```

### Which to use, while loop or for...in loop?

Which type of loop should you use?

A **while** loop is more general. It can be used to handle repetition of a block of code a given number of times and also to handle user controlled repetitions, e.g., when the number of times the loop is executed depends on the user input (or on a condition which depends on a random number).

A **for...in range()** loop is more compact and it is used for repeating a block of code a given number of times. It is useful for processing a block of code for a sequence of values.

### Summary

In a Python program:

- the `Python range()` function is used to define a sequence of values
- a **for...in range()** loop can be used to implement counter-controlled repetition
- a **for...in range()** loop can be converted into a while loop
- a **for...in range()** loop has the following syntax:
  ```python
  for a_variable in range( ):
      statement1
      statement2
      ...
  ```

Examples of Python features used in this lecture

```python
def get_divisor_sum(number):
    div_sum = 1
    middle_num = number // 2
    for num_to_check in range(2, middle_num + 1):
        if number % num_to_check == 0:
            div_sum = div_sum + num_to_check
    return div_sum

def fun_stuff():
    total = 0
    for number in range(9, 20):
        if number % 2 == 0 or number % 3 == 0:
            total += 1
    print(total)
```