At the end of this lecture, students should:

- understand the Python range() function and use it to define a sequence of values
- understand the for...in loop structure used with the range() function
- be able to define a for...in loop to implement counter-controlled repetition
- be able to convert a for...in loop (with a range function) into a while loop and vice versa

The Python range() function

- The Python range() function defines a sequence of integer values within a 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:
- \texttt{range(1, 10, 2)} defines the sequence 1, 3, 5, 7, 9
- \texttt{range(5, 20, 6)} defines the sequence 5, 11, 17
- \texttt{range(14, 4, -3)} defines the sequence 14, 11, 8, 5
- \texttt{range(0, 7, 1)} defines the sequence 0, 1, 2, 3, 4, 5, 6
The Python `range()` function continued

```python
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
  - `range(0, 4)` defines the sequence 0, 1, 2, 3
- 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 does NOT print the defined sequence of integers, i.e., `print(range(6))` does NOT print the sequence 0, 1, 2, 3, 4, 5

---

Iteration – for...in loops

- The following while loop executes an exactly 100 times (for count = 0 to count = 99). The variable count controls the number of times the loop body is executed.

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

- The `for...in range(...)` loop can provide a compact structure for counter-controlled type of loops.

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

Note that in the `for...in range(0, 100)` loop on the previous slide the name used for the loop variable can be any identifier. The following two loops

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

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

behave in exactly the same way.

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

```python
for number in range(0, 5):
    print(number)
```

```python
for number in range(0, 5):
    print(number * 5)
```
Give the output

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

def main():
    show_output()
main()
```

Complete the loops

- Complete the for...in loop so that the output is:
  ```python
  7 10 13 16 19 22
  ```

  ```python
  for number in [7, 10, 13, 16, 19, 22]:
      print(number, end = " ")
  print()
  ```

- Complete the for...in loop so that the output is:
  ```python
  30 25 20 15 10 5 0 -5 -10
  ```

  ```python
  for value in [30, 25, 20, 15, 10, 5, 0, -5, -10]:
      print(value, end = " ")
  print()
  ```

Complete the function

- An amount doubles each year. Using a for...in loop complete the `double_each_year()` function which prints the growth of the parameter value, `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.

  ```python
def double_each_year(start_amt, num_years):
    print_start = True
    print("
    for y in range(1, num_years + 1):
        if print_start:
            print("
            print_start = False
            print("
        else:
            print("
        start_amt = start_amt * 2
        print(start_amt, end = " ")
    print()

def main():
    print_start = True
    print("
    for y in range(1, 6):
        if print_start:
            print("
            print_start = False
            print("
        else:
            print("
        start_amt = start_amt * 2
        print(start_amt, end = " ")
    print()
main()
```

- Using a for...in 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:

  ```python
  print_series(2, 8)
  ```

  ```python
def print_series(start_num, how_many):
    for y in range(1, how_many + 1):
        print_series = True
        print("
        for n in range(1, y + 1):
            if print_series:
                print(n, end = " ")
                print_series = False
            else:
                print(n, end = " ")
        start_num = start_num + n
        print(start_num)

def main():
    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
# Counter-controlled while loop
count = 0
while count < 100:
    print("Programming is fun!")
    count = count + 1

# for...in range(...) loop
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.

### 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)
```

### Same output?

- Do the following two loops give the same output? If not, what is the difference in output and what is the change which needs to be made if I would like the output to be the same?

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

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

### Complete the function

- A perfect number is an integer that is equal to the sum of its divisors (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
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()
```

28 is a perfect number
54 is NOT a perfect number
496 is a perfect number
Complete the function

* The `get_series_sum()` function returns the sum of the given number of terms of the series:

\[
\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots
\]

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):
    # implementation
```

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 section of code any number of times and to handle user controlled repetitions, e.g., executing a piece of code a known size or 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 particularly useful for processing a sequence of values one by one.

Examples of Python features used in this lecture

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