Lecture 12:
for loops, continued; introduction to functions
Announcements

• In Wednesday’s lecture: time for review and questions.

• Exam material: range(functions)
  • that is, 0 up to but not including writing your own functions
Goals

• Get practice using for loops and the range function.

• Know the syntax for defining your own functions.

• Know how to define and use functions that take no arguments and return no values.

• Know how to define use parameters to refer to the input arguments of a function.
The **for** statement: syntax

```plaintext
for var_name in sequence:
    codeblock
```

- **for keyword**
- **in keyword**
- **a variable name**
- **a sequence**: either a list or a call to range

An indented **code block**: one or more statements to be executed **for each** iteration of the loop.
Sequences in Python: Lists

```python
for color in ["red", "green", "blue"]:
    print(color)
```

This is a list: an ordered collection of values. Much more on these later.

This code prints:
red
green
blue
The **for** statement: behavior

```python
for color in ["red", "green", "blue"]:  
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:  

red  
green  
blue

In *each* iteration, the loop variable (*color*) takes on a *different* value from the sequence:

("red", then "green", then "blue")

**Notice:** the loop variable gets updated **automatically** after each iteration!
Sequences in Python: the `range` function

`range(a)`: from 0 up to but not including `a`

```python
for i in range(5):
    print(i, end=" ")
```
prints: 0 1 2 3 4

`range(a, b)`: from `a` up to but not including `b`

```python
for i in range(2, 5):
    print(i, end=" ")
```
prints: 2 3 4

`range(a, b, c)`: sequence from `a` up to but not including `b`
counting in `c`

```python
for i in range(1, 8, 3):
    print(i, end=" ")
```
prints: 1, 4, 7
More on range

```python
for i in range(5):
    print(i, end=" ")
```
prints: 0 1 2 3 4

```python
for i in range(2, 5):
    print(i, end=" ")
```
prints: 2 3 4

```python
for i in range(1, 8, 3):
    print(i, end=" ")
```
prints: 1, 4, 7

Exercise: How many elements are in `range(n)`?

A. 0  
B. n-1  
C. n  
D. 10
More on range

```python
for i in range(5):
    print(i, end=" ")
# prints: 0 1 2 3 4

for i in range(2, 5):
    print(i, end=" ")
# prints: 2 3 4

for i in range(1, 8, 3):
    print(i, end=" ")
# prints: 1, 4, 7
```

**Exercise:** How many elements are in `range(a, b)`?

A. a-b
B. b-a-1
C. b-a+1
D. b-a
More on `range`

```python
for i in range(5):
    print(i, end=" ")  # prints: 0 1 2 3 4

for i in range(2, 5):
    print(i, end=" ")  # prints: 2 3 4

for i in range(1, 8, 3):
    print(i, end=" ")  # prints: 1, 4, 7
```

**Exercise:** How many elements are in `range(a, b, c)`?

Suggestion: try working this out
Today’s Quiz

- 3 minutes
Today’s Quiz

• 3 minutes

• Working with a neighbor: do your answers agree? (2 minutes)
A question about for loops

for value in [1, 16, 4]:
    print(value)
    value = value * 10

(for_quirk.py)
Functions, Revisited

• We’ve been using functions since “Hello, World!”:
  
  ```python
  print("Hello, World!")
  ```

• Built-in functions so far:
  
  ```python
  print, input, type
  ```

• We can import more functions:
  
  ```python
  import math
  import turtle
  math.sqrt(4)
turtle.Turtle()
  ```
Functions, Revisited

What **is** a function, anyway?
It’s a chunk of code with a name.
• It *may* take **arguments** as input
• It *may* do something that has an effect
• It *may* return a value

```python
print("Hello world")
```

**Input(s):**
• 0 or more values
• (optional) sep and end keywords

**Return value:**
• none

**Effects:** prints arguments to the screen, with given separator and end
Functions, Revisited

What \textbf{is} a function, anyway?

It’s a chunk of code with a name.
• It \textit{may} take \texttt{arguments} as input
• It \textit{may} do something that has an effect
• It \textit{may} \texttt{return} a value

\textbf{Input(s):}
• none, or
• a string to print as a prompt

\textbf{Return value:}
• the input from the user

\textbf{Effects:} prompts for user input and reads it from the keyboard
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

Input(s):
• a value

Return value:
• the type of the value

Effects: none
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

\[ \text{math.sin}(\text{math.pi}/2) \]

Input(s):
• a number

Return value:
• the sine of the value

Effects: none
Functions, Revisited

What is a function, anyway?
It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

Input(s):
• a number

Return value:
• none

Effects: moves the turtle forward by the given number of units
Functions, Revisited

What is a function, anyway?

• So far we’ve treated functions as “black boxes”, code someone else wrote that does stuff for us.
• All we know are the inputs, effects, and return value.
• We don’t know how it’s done.

Input(s) → [Black Box] → Return value
(Effects)

This is a great situation to be in!

A bunch of (complicated) stuff is wrapped up in a nice, easy-to-use package.
What if

You want a nice easy-to-use function that does something complicated, but nobody else has written it for you…

Soon, you will have the power to write your own functions.
Writing Functions: Syntax

```python
def name(parameters):
    statements
```

Two important questions:
1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?

Let’s dodge these questions for a moment…
Functions: the simplest kind

No arguments, no return value:

```python
def name():
    statements
```

Example:

```python
def print_hello():
    print("Hello, world!")
```
Demo: Function to print a rectangle of # symbols

Input(s):
- none

Return value:
- none

print_rectangle

Effects: prints a 2x50 rectangle of #s to the screen
Demo: Function to print a rectangle of # symbols

- executing a def statement (function definition) has no effect except defining that function.

- after it is defined, a function can be used whenever and wherever in the program.

- modify to ask user what character to print