CSCI 141

Lecture 10:
Modules, random, loops loops loops loops, range
CSCI 141

Lecture 10:
Modules, random, loops loops loops loops, range

fo(u)r loops, get it?
Happenings

CS Mentors Workshop: COMMAND THE LINE!
Wednesday (today!) 10/16 - 4pm CF 165
Announcements
Announcements

• A3 is out! Due next Tuesday.
Announcements

• A3 is out! Due next Tuesday.
  
  • Start early so you have plenty of time to study for…
Announcements

• A3 is out! Due next Tuesday.
  • Start early so you have plenty of time to study for…
• The midterm exam is a week from Friday!
Announcements

• A3 is out! Due next Tuesday.
  • Start early so you have plenty of time to study for…

• The midterm exam is a week from Friday!
  • Covers material through Monday.
Study Tips
Study Tips

Reading is not enough: **solve problems.**
Study Tips

Reading is not enough: **solve problems**.

- **Goals** slides: can you do these things? Try and see.
Study Tips

Reading is not enough: **solve problems**.

- **Goals** slides: can you do these things? Try and see.

- **Terminology**: be able to discuss the meaning of all words that appear in **blue** in the slides
Study Tips

Reading is not enough: **solve problems**.

- **Goals** slides: can you do these things? Try and see.

- **Terminology**: be able to discuss the meaning of all words that appear in blue in the slides

- **Socrative questions**: make sure you know how to solve them. Then, try code in Thonny or compare answers with your peers.
Study Tips

Reading is not enough: solve problems.

- **Goals** slides: can you do these things? Try and see.

- **Terminology**: be able to discuss the meaning of all words that appear in blue in the slides.

- **Socratic questions**: make sure you know how to solve them. Then, try code in Thonny or compare answers with your peers.

- **Demo code**: solve the same problem without looking at my code.
Study Tips

Reading is not enough: **solve problems**.

- **Goals** slides: can you do these things? Try and see.

- **Terminology**: be able to discuss the meaning of all words that appear in **blue** in the slides.

- **Socrative questions**: make sure you know how to solve them. Then, try code in Thonny or compare answers with your peers.

- **Demo code**: solve the same problem without looking at my code.

- **QOTDs**: still available on Canvas - make sure you know how to solve them.
Study Tips

Reading is not enough: **solve problems**.

- **Goals** slides: can you do these things? Try and see.

- **Terminology**: be able to discuss the meaning of all words that appear in **blue** in the slides

- **Socrative questions**: make sure you know how to solve them. Then, try code in Thonny or compare answers with your peers.

- **Demo code**: solve the same problem without looking at my code.

- **QOTDs**: still available on Canvas - make sure you know how to solve them.

- **Exercises** from the eBook
Study Tips

Reading is not enough: **solve problems.**

- **Goals** slides: can you do these things? Try and see.

- **Terminology**: be able to discuss the meaning of all words that appear in **blue** in the slides

- **Socrative questions**: make sure you know how to solve them. Then, try code in Thonny or compare answers with your peers.

- **Demo code**: solve the same problem without looking at my code.

- **QOTDs**: still available on Canvas - make sure you know how to solve them.

- **Exercises** from the eBook

A study guide including sample coding questions is coming later this week.
Goals

• Know how to import a module and call its functions
  • Know how to generate random numbers using the random module's randint function.
  • Know how to find the documentation for a module and its functions to learn what they do.

• Know the syntax and behavior of the for statement (for loop)

• Know how to use the range function in the header of a for loop.
Last time:
the \texttt{while} statement

Not so different from an \texttt{if} statement:

\begin{itemize}
\item \texttt{while} keyword
\item a boolean expression (the \texttt{condition})
\item a colon : \\
\texttt{while} \ [condition]: \\
[\textit{indented code block}]
\end{itemize}

an indented \texttt{code block}: one or more statements to be executed \texttt{while} the boolean expression evaluates to True
The **while statement:**

**Semantics (Behavior)**

**If statement:**
1. Evaluate the condition
2. If true, execute body (code block), then continue on.

**While statement:**
1. Evaluate the condition
2. If true, execute body, otherwise skip step 3 and continue on.
3. Go back to step 1
The **while statement**: A Working Example

# print account balance after each # of five years:
balance = 100.0 # starting balance
year = 1
while year <= 5:
    balance = balance + (0.02 * balance)
    print(balance)
    year = year + 1

Terminology notes:
• the line with **while** and the condition is the **loop header**
• the code block is the **loop body**
• the entire construct (header and body) is a **while statement**
• usually people call them **while loops** instead
What is the output of the following code?

```python
count = 10
while count < 21:
    print(count, end=" ")
    count += 3
```
What are the values of $m$ and $n$ after this code is executed?

```python
n = 12345
m = 0
while n != 0:
    m = (10 * m) + (n % 10)
    n //= 10
```
What can you do with while loops?

- Anything you can write code to do!
- Not just counting.
Demo: Not just counting
Demo: Not just counting

- `sum_inputs.py`:
  - sum user-provided positive numbers until a negative number is entered
Other Peoples’ Code

We’ve already used code other people wrote by calling built-in Python functions:

• `print`, `input`, `type`

Built-in functions are special because they’re always available.

Many other functions exist in the Python Standard Library, which is a collection of modules containing many more functions.
Other Peoples’ Code

An example: I want to generate a random integer between 0 and 10.

```python
import random
```
An example: I want to generate a random integer between 0 and 10.

I don’t know how to do this.

```
import random
```
Other Peoples’ Code

An example: I want to generate a random integer between 0 and 10.

I don’t know how to do this.

Someone who does has written some functions for me. They live in the `random` module:

```python
import random
```
Other Peoples’ Code

An example: I want to generate a random integer between 0 and 10.

I don’t know how to do this.
Someone who does has written some functions for me. They live in the `random` module:

```python
import random
```

I could go look at the source code…
Other Peoples' Code

An example: I want to generate a random integer between 0 and 10.

```python
import random

# def randrange(self, start, stop=None, step=1, _int=int):


    """Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.
    """

    # This code is a bit messy to make it fast for the
    # common case while still doing adequate error checking.
    istart = _int(start)
    if istart != start:
        raise ValueError("non-integer arg 1 for randrange()")
    if stop is None:
        if istart > 0:
            return self.__randbelow(istart)
        raise ValueError("empty range for randrange()")

    # stop argument supplied.
    istop = _int(stop)
    if istop != stop:
        raise ValueError("non-integer stop for randrange()")

    width = istop - istart
    if step == 1 and width > 0:
        return istart + self.__randbelow(width)
    if step == 1:
        raise ValueError("empty range for randrange() (%d, %d, %d)" % (istart, istop, width))

    # Non-unit step argument supplied.
    istep = _int(step)
    if istep != step:
        raise ValueError("non-integer step for randrange()")
```
Other Peoples’ Code

An example: I want to generate a random integer between 0 and 10.

I don’t know how to do this.

Someone who does has written some functions for me. They live in the random module:

```python
import random

num = random.randint(0, 10)
```

I could go look at the source code… but I’d rather just use their functions without knowing how they work.
import random
num = random.randint(0, 10)

Two questions:

1. What is this syntax about?
2. How do I know what the function does?
Using Modules: Syntax

The Python Standard Library is a collection of modules containing many more functions.

To use functions in a module, you need to import the module using an import statement:

```python
import module
```

By convention, we put all import statements at the top of programs.
Using Modules: Syntax

The Python Standard Library is a collection of modules containing many more functions.

To use functions in a module, you need to import the module using an import statement:

```
import module
```

(replace the in this font with the specific module name)

By convention, we put all import statements at the top of programs.
Using Modules: Syntax

Once you’ve imported a module:

```python
import random
```

you can call functions in that module using the following syntax:

```python
random.randint(0, 10)
```
Using Modules: Syntax

Once you’ve imported a module:

```python
import random
```

you can call functions in that module using the following syntax:

```python
random.randint(0, 10)
```
Using Modules: Syntax

Once you’ve imported a module:

```python
import random
```

you can call functions in that module using the following syntax:

```python
random.randint(0,10)
```
Using Modules: Syntax

Once you’ve imported a module:

```python
import random
```

you can call functions in that module using the following syntax:

```python
random.randint(0, 10)
```

- **Module name**
- **Dot**
- **Function call (the usual syntax)**
Other Peoples’ Code

```python
import random
num = random.randint(0, 10)
```

Two questions:

1. What is this syntax about?
2. How do I know what the function does?
Other Peoples’ Code

```
import random
num = random.randint(0, 10)
```

Two questions:

1. What is this syntax about?

2. How do I know what the function does?

Read about it in the Python documentation.

My approach, in practice:
1. Google “python 3 <whatever>”
2. Make sure the URL is from python.org and has version python 3.x
The math module has useful stuff!

You can read about it in the documentation.

logarithms, trigonometry, ...

Modules can also contain values:

```python
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```
You try it out:

Write a program to compute and print the average of 100 random numbers between 0 and 10.
Hot take: for some tasks, while loops are annoying.
Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do someething() 10 times”
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some\_thing( ) 10 times”

• With a while loop you need to:
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some\_thing() 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:
    some_thing()
    i += 1
```
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do `some_thing()` 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:
    # I don’t even care about i,
    some_thing()  # it’s just bookkeeping!
    i += 1
```
Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `some_thing()` 10 times”

- With a while loop you need to:
  ```python
i = 0
while i < 10:
    some_thing()
    i += 1
```

- Wouldn’t it be great if we could:
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some\_thing() 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:
    some\_thing()
    i += 1
```

• Wouldn’t it be great if we could:

```python
do 10 times:
    some\_thing()
```
Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do some\_thing() 10 times”

- With a while loop you need to:
  
  ```python
  i = 0
  while i < 10:
      some\_thing()
      i += 1
  ```

- Wouldn’t it be great if we could:
  
  ```python
  do 10 times:
      some\_thing()
  ```

We (almost) can! Using for loops.
The **for statement**: syntax

```python
for var_name in sequence:
    codeblock
```
The **for statement**: syntax

```python
for keyword
  for var_name in sequence:
    codeblock
```
The **for statement**: syntax

```
for var_name in sequence:
    codeblock
```
The **for** statement: syntax

```plaintext
for var_name in sequence:
    codeblock
```
The **for** statement: syntax

```
for var_name in sequence:
    codeblock
```
The **for** statement: syntax

`for` keyword → **var_name** → `in` keyword → `sequence:`

An indented **code block**: one or more statements to be executed **for each** iteration of the loop.
The **for** statement: syntax

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

- **for** keyword
- **var_name** (a variable name)
- **in** keyword
- **sequence** (a sequence)
- **colon**
- An indented **code block**: one or more statements to be executed **for each** iteration of the loop
The **for statement**: syntax

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

- **for keyword**
- **var_name**
- **in keyword**
- **sequence**
- **colon**

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 code prints:
red
green
blue
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)
```

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
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:    In each iteration, the loop variable
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: 

```
In each iteration, the loop variable (color)
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:
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")
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: Ranges

Lists are great if you have a list of things, but what about:
Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do some_thing() 10 times”? 

Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do some_thing() 10 times”? ugh.
Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do some\_thing() 10 times”? ugh.

```python
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    some\_thing()
```
Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do some_thing() 10 times”? ugh.

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    some_thing()
```

New function to the rescue: `range` makes it easy to generate lists like this.
Sequences in Python: Ranges

```python
for i in range(5):
    print(i)
```

This code prints:

0
1
2
3
4
Sequences in Python: Ranges

```python
for i in range(5):
    print(i)
```

This code prints:

0
1
2
3
4

The `range` function returns a sequence of integers.
Sequences in Python: Ranges

```python
for i in range(5):
    print(i)
```

This code prints:

0
1
2
3
4

The `range` function returns a sequence of integers.

Not technically a list, but acts like one: more on this later.
Sequences in Python: the `range` function
Sequences in Python: the `range` function

```python
for i in range(5):
    print(i, end=" ")
```

prints: 0 1 2 3 4
Sequences in Python: the \texttt{range} function

\texttt{range}(a): from 0 \textit{up to but not including} a

\begin{verbatim}
for i in range(5):
    print(i, end=" ")
\end{verbatim} prints: 0 1 2 3 4
Sequences in Python: the `range` function

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

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
```
Sequences in Python:

the `range` function

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

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

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

Sequences in Python:

the **range** function

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

```  
for i in range(5):
    print(i, end=" ")  
```

prints: 0 1 2 3 4

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

```  
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
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 *increments* of `c`

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

The \texttt{range} function returns a \texttt{sequence} of integers.

It’s not technically a \texttt{list}: \texttt{print(range(4))} does not print \texttt{[1, 2, 3]}

To turn the range into a list (e.g., to print it), we can use the \texttt{list} function:

\begin{verbatim}
list(range(2, 5)) => [2, 3, 4]
\end{verbatim}
Range function: Demo

- demo in shell
  - one, two, and three argument versions
- ranges.py - poll questions
Range function: Demo
Back to for loops...

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

a variable name

for keyword

in keyword

an indented code block: one or more statements to be executed for each iteration of the loop
Back to **for** loops...

- **for** keyword
- **in** keyword
- **var_name**
- **sequence**
- **codeblock**

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
while loops are annoying.
while loops are annoying.

- Often, you want: “Do some\_thing() 10 times”
while loops are annoying.

• Often, you want: “Do some\_thing() 10 times”

• With a while loop you need to:
while loops are annoying.

- Often, you want: “Do `some_thing()` 10 times”
- With a while loop you need to:

```python
i = 0
while i < 10:
    some_thing()
    i += 1
```
while loops are annoying.

- Often, you want: “Do some\_thing() 10 times”

- With a while loop you need to:

```python
i = 0
while i < 10:  # I don’t even care about i, it’s just bookkeeping!
    some\_thing()
    i += 1
```
while loops are annoying.

• Often, you want: “Do `some_thing()` 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:  # I don’t even care about i,
    some_thing()  # it’s just bookkeeping!
    i += 1
```

• Wouldn’t it be great if we could:
while loops are annoying.

• Often, you want: “Do `some_thing()` 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:  # I don’t even care about i,
    some_thing()  # it’s just bookkeeping!
    i += 1
```

• Wouldn’t it be great if we could:

```python
for i in range(10):
    some_thing()
```
while loops are annoying.

• Often, you want: “Do `some_thing()` 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:
    some_thing()
    i += 1
```

• Wouldn’t it be great if we could:

```python
for i in range(10):
    some_thing()
```

We can!
Revisiting Repetition

\texttt{for var\_name in sequence:}
\texttt{codeblock}

- balance3.py - rewrite yearly bank account balance with a for loop

- Average of 100 random numbers
while vs for

**Task:** Generate and print random integers between 1 and 10 (inclusive) until one of the random numbers exceeds 8.

Would you use a for loop or a while loop?
while vs for

Task: Ask the user for a number (n), then print 100 random numbers between 0 and n.

Would you use a for loop or a while loop?
while vs for

**Task:** Sum the numbers from 1 to 340, leaving out those divisible by 5.

Would you use a for loop or a while loop?
A1 debrief