CSCI 141

Lecture 9:
Repetition: Repetition, the while statement,
Repetition, Repetition, Repetition, Modules
Announcements
Announcements

- A2 is due tomorrow night!
Announcements

• A2 is due tomorrow night!

• Q2: Write a program called quadratic.py that takes three floating-point numbers as command line arguments(!)
Announcements

• A2 is due tomorrow night!

  • Q2: Write a program called quadratic.py that takes three floating-point numbers as **command line arguments**(!)

• A3 will be out tomorrow.
Announcements

• A2 is due tomorrow night!
  • Q2: Write a program called quadratic.py that takes three floating-point numbers as command line arguments(!)

• A3 will be out tomorrow.
  • Due next Tuesday 10/22
Announcements

• A2 is due tomorrow night!
  
  • Q2: Write a program called quadratic.py that takes three floating-point numbers as command line arguments(!)
  
• A3 will be out tomorrow.
  
  • Due next Tuesday 10/22
  
• Midterm exam is a week from Friday!
Announcements

• A2 is due tomorrow night!
  
  • Q2: Write a program called quadratic.py that takes three floating-point numbers as command line arguments(!)

• A3 will be out tomorrow.
  
  • Due next Tuesday 10/22

• Midterm exam is a week from Friday!
  
  • Notes on how to study coming up next lecture.
Goals

• Understand the syntax and behavior of the while statement (also known as while loop).

• Know how to use in-place operators: +=, -=, etc.

• Know how to import a module and call its functions
  
  • Know how to generate random numbers using the random module's randrange function.
Last time: if statements

if keyword | a boolean expression (the condition) | a colon:

```
if isRaining:
    print("You should wear a raincoat!")
```

an indented code block: one or more statements to be executed if the boolean expression evaluates to True
if isRaining and not isWindy:
    print("Bring an umbrella!")
elif isRaining and isWindy:
    print("Wear a raincoat!")
else:
    print("No rain gear needed!")
Last Time: Chained Conditionals

**elif keyword**

```python
if isRaining and not isWindy:
    print("Bring an umbrella!")
elif isRaining and isWindy:
    print("Wear a raincoat!")
else:
    print("No rain gear needed!")
```

- *an indented code block to be executed if:
  - **none** of the above conditions was True
  - **and** this `elif`’s condition is True*
if isRaining and not isWindy:
    print("Bring an umbrella!")
elif isRaining and isWindy:
    print("Wear a raincoat!")
else:
    print("No rain gear needed!")
Last Time: Chained Conditionals

`elif` keyword

```python
if isRaining and not isWindy:
    print("Bring an umbrella!")
elif isRaining and isWindy:
    print("Wear a raincoat!")
else:
    print("No rain gear needed!")
```

An indented code block to be executed if:
- none of the above conditions was True
- and this `elif`’s condition is True

(this behaves exactly like nesting an if inside each else)

An indented code block to be executed if the none of the above conditions was true
if isRaining and not isWindy:
    print("Bring an umbrella!")
elif isRaining and isWindy:
    print("Wear a raincoat!")
else:
    print("No rain gear needed!")
QOTD

Program 1:

```python
if (num_tacos == 32):
    print("32 tacos")
elif (num_tacos < 32):
    print("Too few tacos")
elif (num_tacos == 32):
    print("32 tacos")
elif (num_tacos % 5 == 0):
    print("Oh yes, tacos!")
else:
    print("Too many tacos")
```

Give the **smallest positive integer** value for the variable num_tacos such that the three programs print **exactly** the same thing when they are executed.
Program 2:

```python
if (num_tacos == 32):
    print("32 tacos")
if (num_tacos < 32):
    print("Too few tacos")
if (num_tacos == 33):
    print("33 tacos")
if (num_tacos % 5 == 0):
    print("Oh yes, tacos!")
else:
    print("Too many tacos")
```

Give the smallest positive integer value for the variable num_tacos such that the three programs print exactly the same thing when they are executed.
QOTD

Program 3:

```python
if (num_tacos == 32):
    print("32 tacos")
else:
    if (num_tacos < 32):
        print("Too few tacos")
    else:
        if (num_tacos == 34):
            print("34 tacos")
        else:
            if (num_tacos % 5 == 0):
                print("Oh yes, tacos!")
            else:
                print("Too many tacos")
```

Give the **smallest positive integer** value for the variable `num_tacos` such that the three programs print **exactly** the same thing when they are executed.
Today: Repetition

• So far, we’ve seen how to:
  • Print things to the screen and replace your calculator
  • Represent complicated boolean expressions and execute different code based on their truth values.

• So far we *haven’t* seen how to:
  • Do anything that you couldn’t do yourself, given pencil and paper and a few minutes to step through the code.
Motivation

Anyone really good at tongue twisters?

Pad kid poured curd pulled cod.
Pad kid poured curd pulled cod.
Pad kid poured curd pulled cod.
Pad kid poured curd pulled cod.
Pad kid poured curd pulled cod.
Pad kid poured curd pulled cod.

This is (according to MIT psychologists*) the hardest known tongue twister.

Fact: humans are bad (or at least slow) at performing repetitive tasks.

*Stefanie Shattuck-Hufnagel et al., 2013
Motivation

**Fact:** humans are **bad** (or at least slow) at performing repetitive tasks.

[https://www.naturalreaders.com/online/](https://www.naturalreaders.com/online/)

**Fact:** computers are good (or at least fast) at performing repetitive tasks.
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance after five years?
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

\[
\text{balance} = 100.00
\]
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

balance = 100.00
balance = balance + (0.02 * balance)
print(balance)  # year 1
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

```python
balance = 100.00
balance = balance + (0.02 * balance)
print(balance)  # year 1
balance = balance + (0.02 * balance)
print(balance)  # year 2
```
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

```python
balance = 100.00
balance = balance + (0.02 * balance)
print(balance)  # year 1
balance = balance + (0.02 * balance)
print(balance)  # year 2
balance = balance + (0.02 * balance)
print(balance)  # year 3
```
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

```
balance = 100.00
balance = balance + (0.02 * balance)
print(balance)  # year 1
balance = balance + (0.02 * balance)
print(balance)  # year 2
balance = balance + (0.02 * balance)
print(balance)  # year 3
balance = balance + (0.02 * balance)
print(balance)  # year 4
```
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

```python
balance = 100.00
balance = balance + (0.02 * balance)
print(balance)  # year 1
balance = balance + (0.02 * balance)
print(balance)  # year 2
balance = balance + (0.02 * balance)
print(balance)  # year 3
balance = balance + (0.02 * balance)
print(balance)  # year 4
```

uh oh… my font is getting small
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

```plaintext
balance = 100.00
balance = balance + (0.02 * balance)
print(balance)  # year 1
balance = balance + (0.02 * balance)
print(balance)  # year 2
balance = balance + (0.02 * balance)
print(balance)  # year 3
balance = balance + (0.02 * balance)
print(balance)  # year 4
balance = balance + (0.02 * balance)
print(balance)  # year 5
```
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for five years?

```
balance = 100.00
balance = balance + (0.02 * balance)
print(balance) # year 1
balance = balance + (0.02 * balance)
print(balance) # year 2
balance = balance + (0.02 * balance)
print(balance) # year 3
balance = balance + (0.02 * balance)
print(balance) # year 4
balance = balance + (0.02 * balance)
print(balance) # year 5
```

argh, ok, done.
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for 500 years?

An extremely common task: do the same thing over and over again, or do the same processing on many pieces of data.
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for 500 years?

An extremely common task: do the same thing over and over again, or do the same processing on many pieces of data.
Motivation

Suppose you have a starting bank account balance of $100.00, and your account earns 2% interest each year.

What is your balance each year for 500 years?

An extremely common task: do the same thing over and over again, or do the same processing on many pieces of data.
Motivation

Example: Convert this 100x100 pixel image to grayscale ("black-and-white").
Motivation

Example: Convert this 100x100 pixel image to grayscale ("black-and-white").
Motivation

Example: Convert this 100x100 pixel image to grayscale ("black-and-white").
Motivation

Example: Convert this 100x100 pixel image to grayscale (“black-and-white”).
Motivation

Example: Convert this 100x100 pixel image to grayscale ("black-and-white").

10,000 pixels, same calculation:

\[
grey = 0.29 \times red + 0.59 \times green + 0.12 \times blue
\]
Python to the rescue: the **while** statement

Not so different from an **if** statement:

```python
if keyword
    a boolean expression (the condition)
    a colon:
    ```
    
    ```python
    if year <= 5:
        balance = balance + (0.02 * balance)
    print(balance)
    ```

an indented **code block**: one or more statements to be executed **if** the boolean expression evaluates to **True**
Python to the rescue: the **while** statement

Not so different from an **if** statement:

```python
while year <= 5:
    balance = balance + (0.02 * balance)
    print(balance)
```

**while** keyword  
**a boolean expression (the condition)**
**a colon**: 

**an indented code block**: one or more statements to be executed **while** the boolean expression evaluates to True
The **while** statement: A Working Example

```python
# 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
```
The **while** statement: A Working Example

```python
# 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 **while** statement: A Working Example

```python
# 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 **while** statement: A Working Example

```python
# 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 **while** statement: A Working Example

```python
# 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**
The **while** statement: A Working Example

```python
# 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
demo: interest

• balance1.py: the tedious way

• balance2.py: the loopy way
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
Doubling to 100

Task: Find how many times you have to double the number 1 to make it larger than 100.
Doubling to 100

**Task:** Find how many times you have to double the number 1 to make it larger than 100.

```python
times = 0
n = 1
while [condition here]:
    n = n * 2
    times = times + 1
print(times, "times!")
```
Task: Find how many times you have to double the number 1 to make it larger than 100.

times = 0
n = 1
while [condition here]:
    n = n * 2
    times = times + 1
print(times, "times!")

Which of the following conditions is correct?
A. times < 100
B. times <= 100
C. n > 100
D. n <= 100
Aside: In-Place Operators

When writing loops (and code in general), you’ll find yourself doing things like this often:

```python
count = count - 1
total = total + n
```

Python has a nice shorthand for this:

```python
count -= 1
total += n
```

Many math operators have an in-place version:

```
+=    -=    /=    //=    %= 
```
Aside: In-Place Operators

When writing loops (and code in general), you’ll find yourself doing things like this often:

```python
count = count - 1
total = total + n
```

Python has a nice shorthand for this:

```python
count -= 1
total += n
```

Many math operators have an in-place version:

```plaintext
+=    -=    /=    //=    %=   
```

[No, Python doesn’t have increment and decrement operators ++ and --]
Demo
• double.py - change to in-place operators

• count.py:
  • Counting up, counting down by an interval

• never.py:
  • Condition never True
  • Condition never False

• input.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.
An example: I want to generate a random integer between 0 and 10.
Other Peoples’ Code

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

I don’t know how to do this.
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...
An example: I want to generate a random integer between 0 and 10.

```python
import random
```

I don't know how to do this.

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

```python
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.
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?
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
```

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 text 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)
```

Module name
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

Function call (the usual syntax)
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

```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?

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 example
Demo

• use of randint in a very simple guessing game
math module

• 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
```
More on import statements

• Import the entire module:

```python
import random
num = random.randint(1, 10)
```

• Import a specific function:

```python
from math import sin
sin0 = sin(0)
```

• Don’t need module name dot notation
• Other `math` functions are not accessible
You try it

**Exercise:** write a program that generates and prints random integers between 1 and 10 (inclusive) until one of the random numbers exceeds 8.

Documentation says:

```python
random.randint(a, b)
```

Return a random integer $N$ such that $a \leq N \leq b$