Repetition: the **while** statement
Goals

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

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

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?
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
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?

\[
\text{balance} = 100.00 \\
\text{balance} = \text{balance} + (0.02 \times \text{balance}) \\
\text{print}(\text{balance}) \ # \ year \ 1
\]
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

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?

```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
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 thing to many different pieces of data.
Motivation

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

10,000 pixels, same calculation:

grey = 0.29 \times \text{red} + 0.59 \times \text{green} + 0.12 \times \text{blue}
Python to the rescue: the **while** statement

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

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

**if** keyword

- a boolean expression (the **condition**)
- a colon:

- 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 keyword

a boolean expression (the condition)

```

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

**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
```

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
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
Code Examples

• balance1.py: the tedious way

• balance2.py: the loopy way
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:

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

[No, Python doesn’t have increment and decrement operators ++ and --]
Demo
Demo

- double.py
  - Count how many times you need to double 1 before it exceeds 100

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

- never.py:
  - Condition never True
  - Condition never False