Try this

- Start with the last four digits of your phone number
- Multiply it by 2
- Add 5
- Multiply 50
- Add 1764
- Subtract the year you were born (e.g., 1995)
- Now, see what the first 4 digits of the resulting number is (your phone number?)
- And see the last 2 digits of the resulting number (your age?)

- Can you write a function that does this?
- And call it a few times?
Announcements

- PS 3 is up and due this Sunday
- First midterm will be in class on Tuesday 10/13
  - Closed book but a sheet (8.5x11) of paper is allowed (both sides)
  - See practice problems in 10/1 lecture notes area
- Try to get help
- Reading assignment for this slide set:
  - Chapters 7, 8, 9.1 of Downey
  - You are welcome to try 9.2-9.4 but not required
- Break around 10:15am
Strings
Reading files
Type: `str`

- Type `str` for strings
  - **Values**: any sequence of characters
  - **Operation(s)**: + (concatenation)

- **String literal**: sequence of characters in quotes
  - Double quotes: “abc3$#@()”, “Hello world!”
  - Single quotes: ‘Hello world!’

- Concatenation can only apply to strings
  - “ab” + “cd” evaluates to “abcd”
  - “ab” + 2 produces an error
  - “ab” + str(2) evaluates to “ab2”
String: text as a value

- **string**: a quoted sequence of characters
  - ‘abc d’ (Python prefers)
  - “abc d” (most languages)

- How to write quotes in quotes?
  - Delineate with “other quote”
  - **Examples**
    - “ ‘ “
    - ‘ ‘ ‘
    - “a’b’c” // third character is b
    - ‘a”bb”c’
Strings are indexed

- \( s = 'abc\ d' \)
- **Index:** 0-based

- Access characters with \[ \] and *slicing*
  - \( s[0] == 'a' \)
  - \( s[4] == 'd' \)
  - \( s[5] \) *causes an error*
  - \( s[0:2] == 'ab' \) *(excludes c)*
  - \( s[2:] == 'c\ d' \)
  - \( s[:2] == 'ab' \)
  - \( s[:] == 'abc\ d' \)

- Called *“string slicing”*
  - \([n:m]\) – from \( n \)-th inclusive to \( m \)-th exclusive
More examples

\[
\begin{align*}
\text{s} &= \text{‘Hello all’} \\
\text{s.index(‘l’)} &= 2 \\
\text{s[2:4]} &= \text{‘ll’} \\
\text{s[:4]} &= \text{‘Hell’} \\
\text{s[4:]} &= \text{‘o all’} \\
\text{s.index(s[7:])]} &= 2
\end{align*}
\]
A string puzzle

- **Given**: variable `student` contains a string with at least two ‘,’s
  
  `student = 'John Doh,History,909-432-3456'`

- **Goal**: extract the substring that comes after the second ‘,’

**Plan**
- Store the index of the first ‘,’ in a variable `k`
- Store the substring from `k` to the end in a variable `tail`
- Extract the substring from ‘,’ to the end in `tail`

**Code**

```python
k = student.index(',,')
tail = student[k+1:]
tail[tail.index(',,')+1:]
```
Other ways to get data from strings

- `s1 in s2`: tests if `s1` is “a part of” `s2`
- `len(s)`: value is number of chars in `s`
- `s1.index(s2)`: position of the 1st instance of `s2` in `s1`
- `s1.count(s2)`: number of times `s2` appears inside `s1`
- `s.strip()`: a copy of `s` with white-space removed from both ends
  `s1.strip(s2)` removes `s2` from both ends of `s1`, if any.
Examples

s = 'abracadabra'
'a' in s == True
s.index('a') == 0
s.count('a') == 5
len(s) == 11
s.strip('a') == 'bracadabr'
' cs40 '.strip() == 'cs40'
s = 'Apple'
s*3 == 'AppleAppleApple'

- A lot more information in Python documentation on www.python.org (see Library Reference, built-in types there)
Traversing strings with `for...in`

- Computations often involve processing a string one character at a time
- They often start with the first character, select each character in turn, do something with it, and continue until the end
- Examples:

```python
hobby = 'ultimate frisbee'
i = 0
while i < len(hobby):
    ch = hobby[i]
    print ch
    i = i + 1
```

```python
hobby = 'ultimate frisbee'
for ch in hobby:
    print ch
```
Exercises

- Write a function named `backward` that takes a string as an argument and prints the letters in the string backward, one per line (Hint: use a `while` loop)

- This time write one named `backward2` that does the same but all on the same line

- This time write one named `backward3` that does the same but every other letter

- This time write one named `backward4` that does the same but returns the string backward rather than printing it to the screen.
def backward(s):
    i = len(s) - 1
    while i >= 0:
        print s[i]
        i = i - 1
    backward('Ultimate')

def backward2(s):
    i = len(s) - 1
    while i >= 0:
        print s[i],
        i = i - 1
backward2('Ultimate')

def backward3(s):
    i = len(s) - 1
    while i >= 0:
        print s[i],
        i = i - 2
    backward3('abc')
print
backward3('abcd')
print
backward3('')

def backward4(s):
    res = '
    for c in s:
        res = c + res
    return res
print backward4('apple')
Review: String concatenation

- We can use + to add (concatenate) two strings
- Try Exercise 8.2 on page 73 of the text
Strings are *immutable* (next)

- Strings cannot be changed in place, e.g., the following is not legal in Python:

  ```python
  sports = 'swimming'
sports[0] = 'S'          # error
  ```

- We can create a new string though

  ```python
  sports = 'S' + sports[1:]  # OK
  ```
Searching in a string

• Write a function named `find` with two parameters (`word` and `letter`) that returns the index of the letter in `word` if found; `-1` otherwise.

• You may use either a `for` loop or a `while` loop (or even recursion); I suggest you try both `for` and `while` to get more practice.

• This time add a third parameter, the index in `word` where it should start looking. Name this function `find2`
def find(word, letter):
    i = 0
    for ch in word:
        if ch == letter:
            return i
        i = i + 1
    return -1

print find('apple', 'e')
print find('apple', 'k')
def find1(word, letter):
    i = 0
    while i < len(word):
        if word[i] == letter:
            return i
        i = i + 1
    return -1

print find1('apple', 'e')
print find1('apple', 'k')
def find2(word, letter, start):
    i = start
    while i < len(word):
        if word[i] == letter:
            return i
        i = i + 1
    return -1

print find2('apple', 'e', 0)
print find2('apple', 'k', 0)
More exercise

- Write a function named `count` with two string parameters (word and letter) which counts and returns the number of times the letter appears in the word
def count(word, letter):
    c = 0
    for ch in word:
        if ch == letter:
            c = c + 1
    return c

print count('appleapple', 'a')
print count('', 'a')
print count('appleapple', 'k')
String methods

- A *method* is similar to a function
  - It takes arguments and returns a value
  - But the syntax is different

- Function syntax: `upper(word)` // there is no function named upper
- Method syntax: `word.upper()` // there is a method though

- Example:

```python
>>> word = 'apple'
>>> new_word = word.upper()
>>> print new_word
APPLE
```
**find method in string**

- Similar to the one we wrote, but more general
  - Can find substrings returning an index, not just characters
  - Can take a second argument, the index where it should start
  - Can take a third argument, the index where it should stop

```
>>> word = 'applepear'
>>> word.find('e')
4
>>> word.find('lep')
3
>>> word.find('lep', 3)
3
```

For other methods see:
https://docs.python.org/2/library/stdtypes.html#string-methods
String comparison

- The relational operators (==, !=, <, >, <=, >=) work on strings

```python
>>> fruit = 'apple'
>>> fruit == 'APPLE'
False
>>> fruit != 'orange'
True
>>> fruit > 'orange'
False
>>> 'kiwi' < 'orange'
True
>>> 'kiwi' <= 'kiwia'
True
>>> 'kiwi' >= 'kiwib'
False
```
Reading a file of words

- To read a file:
  - **First**, open the file
  - **Then**, start reading one line at a time or the entire file all at once
  - **Finally**, close the file when you are done using the file

- **Example:** Let’s read a file named `words.txt` which contains some English words (see what the file contains first)

```python
# cd to the folder where words.txt is and start python on command line
>>> fin = open('words.txt')
>>> fin
<open file 'words.txt', mode 'r' at 0x104671ed0>
>>> fin.readline()
'apple is a fruit
'
>>> line = fin.readline()
>>> line
'bear is an animal
'
>>> line.strip()
'bear is an animal'
>>> fin.read()   # see what it does
```

We will do all this in a script file on the next page
Reading the entire file

- Let's read a file named `words.txt` and count the lines in the file

```python
def display_file(filename):
    fin = open(filename)
    for line in fin:
        print line.strip()
    fin.close()

display_file('words.txt')
```

```python
def file_line_count(fname):
    fin = open(fname)
    count = 0
    for line in fin:
        count = count + 1
    fin.close()
    return count

print file_line_count('words.txt')
```
Reading a line or the entire file

- You can read one line at a time or an entire file all at once as we saw briefly

- See `read_file.py` for more examples
Do these before next class

- Finish reading Chapter 8
  - Try the exercises from the lecture slides
  - Try them without seeing my solutions first and compare yours with mine

- Next topic
  - Lists (chapter 10)