Tuples and Strings

but mostly strings

adapted from material by Mike Scott and Bill Young at the University of Texas at Austin

Tuples

Like a list, a tuple is a sequence. Unlike a list, it is *immutable*.

Once you have created it, you cannot change it.

```
tup = (1, 2, 3, "a")     # Can be heterogeneous, just like lists
print(tup[2])            # Can be indexed
print(tup[1:3])          # Can be sliced
tup[2] = 4               # TypeError
```

A tuple is like an immutable list. Which of the following operations do we think tuples support?

- index
- append
- slicing
- len
- max
- remove
- +
- insert
- *
- comparison
- reverse
- sort
- in
- indexassign

Destructuring

Remember that we have *multiple assignment* syntax in Python

```
a, b = 3, 5
```

We can do a similar thing to *destructure* tuples:

```
tup = (1, 3, 5, 7)
(a, b, c, d) = tup
print(a)
print(b)
print(c)
print(d)
```

Tuples vs Lists

- Tuples can be a little faster than lists (though you should never make this your primary reason for choosing them)
- Tuples can be **safer** if available

```
lst = [1,3,5]
tup = (1,3,5)
mystery_function(lst)
mystery_function(tup)
```

What is the value of lst, tup, after the function calls?

(More about) Strings

Strings are sequences!

Many of the things we learned about sequences will apply.

```
mystring = "Hello!"

for ch in mystring:
  print(ch)

print(ch[3])
print(ch[-2])
```

We even get the same errors!

```
In [1]: lst = [2, 3, 5, 7, 11, 13]

In [2]: s = "whatisthis?"

In [3]: lst[100]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 lst[100]

IndexError: list index out of range

In [4]: s[100]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 s[100]

IndexError: string index out of range
```

Concatenation

Strings can be joined together with the + operator. We can even do += like with numbers.

```
name = input("What is your name? ")
print("Hello " + name)
output = "My user is "
output += name

# Remember that a += x expands to a = a + x
print(output)
```

Strings are Immutable

Like tuples, once strings have been created, they cannot be changed.

```
In [1]: s1 = "Hello"

In [2]: id(s1)
Out[2]: 140240118077168

In [3]: s1 += " human!"

In [4]: print(s1)
Hello human!

In [5]: id(s1)
Out[5]: 140240117404080
```

When it looks like we're modifying a string, we're actually creating a new string (as can be seen here, by the id of s1 changing).

```
s1 = "Hello!"
s1[5] = "?"      # Allowed?
```

One thing to note

In some languages, the individual pieces of a string are a different type (usually known as a "char" or a "byte").

```
22:12:13 ∈ ❯ ocaml
        OCaml version 4.13.1

# let s1 = "Hello!";;
val s1 : string = "Hello!"
# let c = s1.[2];;
val c : char = 'l'
#
```

```
In [1]: s1 = "Hello!"

In [2]: c = s1[3]

In [3]: c
Out[3]: 'l'

In [4]: type(c)
Out[4]: str
```

In Python, this is not the case. The smallest piece of a string is still a string.

String Methods

These methods are used to check if certain properties of the string are true.

Fun fact: these are sometimes known as "predicates" in computer science. It's simply a function that returns True or False.

Method	Description
isalnum()	Does string only contain alphabetical/numerical?
isalpha()	Does string only contain alphabetical characters?
isdigit()	Does string only contain numeric digits?
islower()	Does string consist only of lowercase characters?
isspace()	Does string consist only of whitespace?
isupper()	Does string consist only of uppercase characters?

Let's Write a Program

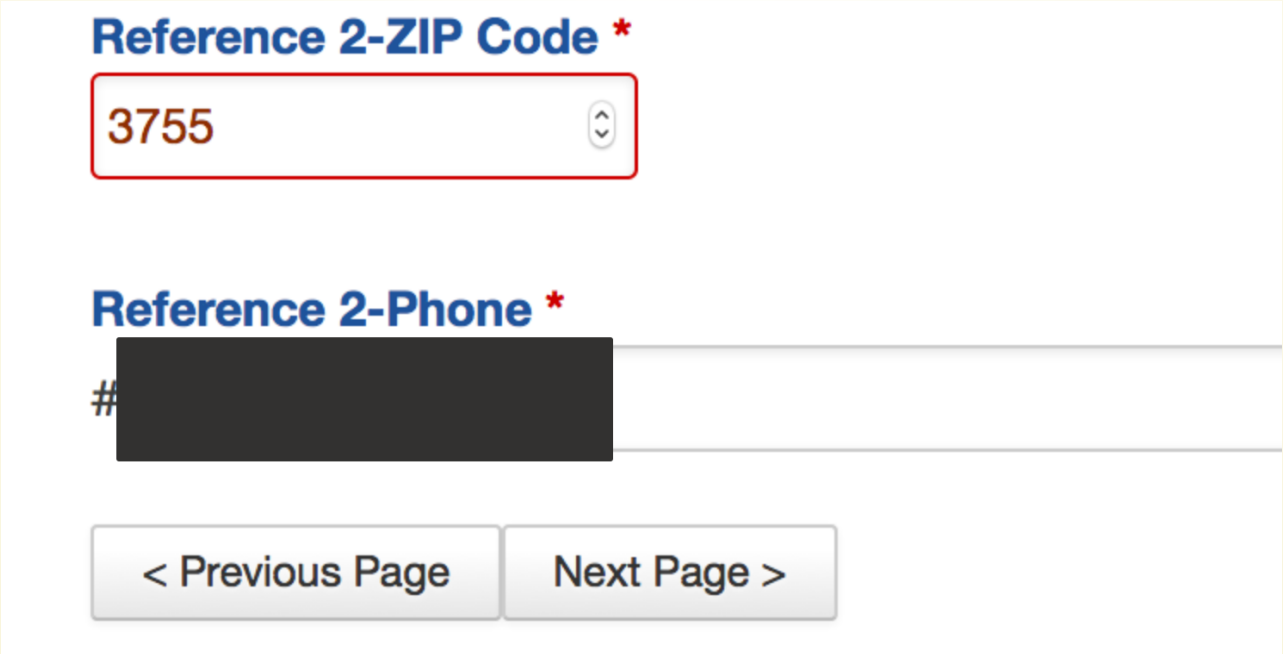
Ask the user to enter a number, then keep bothering them until they actually enter a number.

(Don't use try-except)

Storytime!

One of my relatives lives in the ZIP code 03755.

When I tried to enter this into the form, it told me this was an invalid ZIP code.



Let's Write a Program

Validate a ZIP code **correctly**.

These methods appear to modify the string

Method	Description
lower()	Convert the string to all lowercase.
upper()	Convert the string to all uppercase.
lstrip(ch)	Remove all occurrences of ch from the start of the string (the "left" side of the string)
rstrip(ch)	Remove all occurrences of ch from the end of the string (the "right" side of the string)
strip(ch)	Remove all occurrences of ch from both ends of the string.

If an argument is not provided for any of the strip methods, Python will strip all whitespace.

BE CAREFUL WITH STRING MANIPULATIONS!

```
1  def get_player_names():
2    return ["Anton EgO", "Alfredo Linguini ", "Remy"]
3
4  def player_is_registered(name):
5    return name in get_player_names()
6
7  def main():
8    name = input("What is your player name? ")
9    if player_is_registered(name):
10     print("Welcome " + name)
11   else:
12     print(f"I do not see {name} in my records.")
13
14 main()
```

Smart move: use lower() or upper() to make sure everything is same-cased, and use isalnum() to make sure a string is what you expect.

Search and Splitting

Searching

Sometimes we want to find *substrings* in a string, or to figure out where those substrings are located.

If you want to check if a string starts or ends with a substring, use startswith() or endswith()

```
function is_a_doctor(full_name):
  return full_name.startswith("Dr. ")
```

If you want to know **where** the match occurs, use find(), which returns the index of the start of the match (or -1 if no match is found).

Splitting

Calling s.split(ch) will split a string into multiple strings on the specified character. If no argument is provided, it will split on whitespace.

```
In [1]: s = "Misty, 27, 370, Hello"

In [2]: s.split(',')
Out[2]: ['Misty', ' 27', ' 370', ' Hello']
```

Makes CSV Processing a lot easier!

Write a function which returns a 2D List of strings corresponding to the reading of a CSV