Silly Strings

Hit **Space** to move forward and **Shift + Space** to move backward

What are strings?

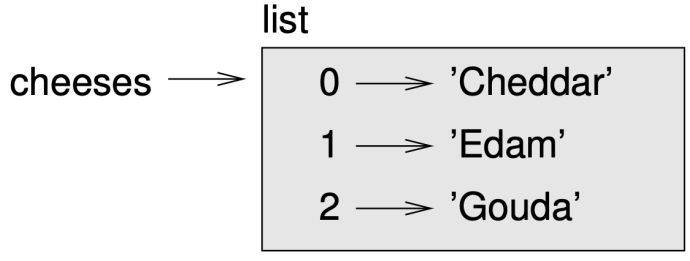
So far, we have used string, integer, float, and boolean values.

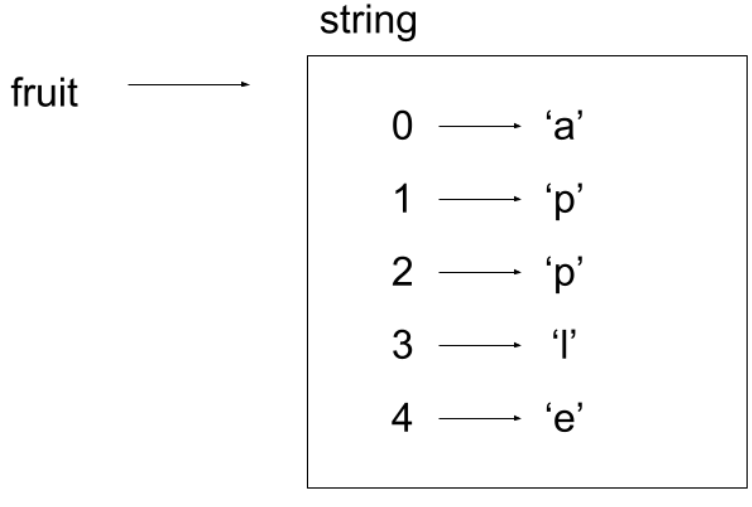
```
bank_name = 'JSeaMorgan Chase & Co.'
principal_amount = 1815
introductory_bonus = 0
annual_interest_rate = 2.9

while True:
    ...
```

But, strings are not like integers, float, and booleans! Each integer, float, or boolean is one value. However, a string is a **sequence**, or ordered collection, of values—characters, to be precise!

Lists are also sequences. Essentially, strings are lists whose elements are characters.





Strings represent textual data

Web applications

- Email
- Password
- Name
- Posts
- Comments

```
email = 'ykim@allegheny.edu'
password = 'illnevertell'
name = 'Maria'
posts = [
    'Today, I went biking.',
    'Look at my salad!',
    'Feeling cute, might delete later.'
]
comments = ['Good luck!', "Let's catch up sometime :)"]
```

Computational biology

Genomic sequence!

```
normal = 'ATGGTGCACCTGACTCCTGAGGAGAAGTCTGCCGTTACT'
sample = 'ATGGTGCACCTGACTCCTGTGGAGAAGTCTGCCGTTACT'
```

How to create a string

Surround text in quotes (single `'` or double `"`--choose your own adventure!).

In [1]:

```
cheese = 'Cheddar'
fruit = "apple"
lyrics = '''I'm a lumberjack, and I'm okay.
    I sleep all night and I work all day.''' # What am I?
print(lyrics)
```

```
I'm a lumberjack, and I'm okay.
    I sleep all night and I work all day.
```

Sometimes, you want to insert dynamic values into static strings

```
# Remember me?
print(f'''
Bank: {bank_name}
Principal amount: ${principal_amount}
Introductory bonus: ${introductory_bonus}
Annual interest rate: {annual_interest_rate}%
Time: {time} years
Accrued amount: ${accrued_amount}
''')
```

How to create a format string

1. Start the string with the letter `f`
2. Then, create the string as usual
3. Within the string, surround expressions that evaluate to the dynamic values you want to insert with curly braces `{}`

In [2]:

```
profession = 'programmer'
lyrics = f'''I'm a {profession}, and I'm okay.
I sleep all night and I work all day.'''
print(lyrics)
```

```
I'm a programmer, and I'm okay.
I sleep all night and I work all day.
```

Accessing a character, or characters, in a string

Sometimes, you want to get a subsection, such as a character or characters, of a string, but not the whole string. A subsection of a string is called a **substring**.

When would you want to access a substring?

To check for known point mutations:

Table 1: Single-Base Mutation Associated with Sickle-Cell Anemia

Sequence for Wild-Type Hemoglobin												
ATG	GTG	CAC	CTG	ACT	CCT	GAG	GAG	AAG	TCT	GCC	GTT	ACT
Start	Val	His	Leu	Thr	Pro	Glu	Glu	Lys	Ser	Ala	Val	Thr
Sequence for Mutant (Sickle-Cell) Hemoglobin												
ATG	GTG	CAC	CTG	ACT	CCT	GTG	GAG	AAG	TCT	GCC	GTT	ACT
Start	Val	His	Leu	Thr	Pro	Val	Glu	Lys	Ser	Ala	Val	Thr

How to access a character in a string

Use the bracket operator `[]` to specify the index of the character.

The first character of a string is at index 0.

In [3]:

```
normal = 'ATGGTGCACCTGACTCCTGAGGAGAAGTCTGCCGTTACT'
nucleotide = normal[19]
print(f'The 20th nucleotide is {nucleotide}.') # Why 20th?
```

```
The 20th nucleotide is A.
```

In [4]:

```
normal = 'ATGGTGCACCTGACTCCTGAGGAGAAGTCTGCCGTTACT'
sample = 'ATGGTGCACCTGACTCCTGTGGAGAAGTCTGCCGTTACT'

# Does patient have sickle-cell anemia?

mutation_location = 19
if sample[mutation_location] != normal[mutation_location]: # Index can be any expression that evaluates to an integer
    print('Sickle-cell mutation found.')
```

```
Sickle-cell mutation found.
```

How to access characters in a string

You can access multiple characters at once by using the slice operator— `[n:m]`.

The slice operator will return characters from index `n` to `m - 1`.

In [5]:

```
title = "Monty Python"
print(title[0:5])
print(title[6:12])
```

```
Monty
Python
```

Strings are immutable

In the Love It or List It module, we learned that you can assign an element, or elements, in a list using bracket notation:

In [6]:

```
maria_lucky_numbers = [7, 128, 23]
print(maria_lucky_numbers)
maria_lucky_numbers[0] = 17
print(maria_lucky_numbers)
```

```
[7, 128, 23]
[17, 128, 23]
```

So, can we assign a character in a string using bracket notation?

In [7]:

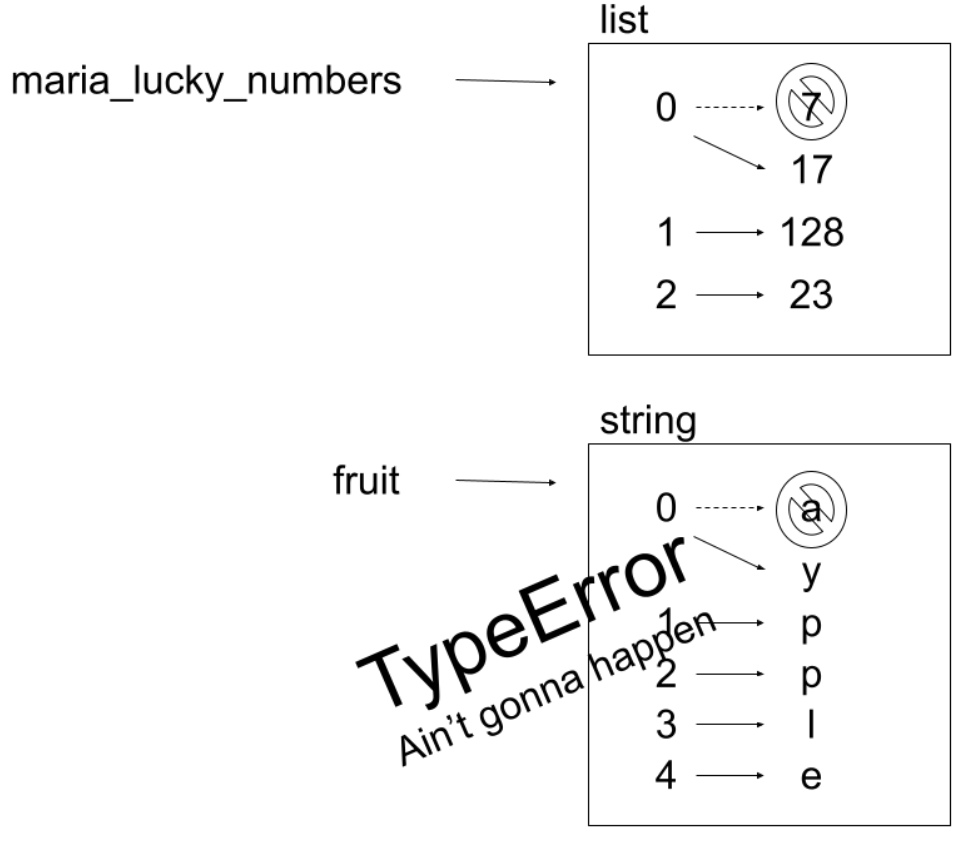
```
fruit = 'apple'
print(fruit)
fruit[0] = 'y'
print(fruit)
```

```
apple
```

```
---------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/bj/bw1mwdzj6vsbs4zf676n7rw80000gq/T/ipykernel_40304/957949467.py in <module>
      1 fruit = 'apple'
      2 print(fruit)
----> 3 fruit[0] = 'y'
      4 print(fruit)

TypeError: 'str' object does not support item assignment
```

No! This is because strings are **immutable**--they cannot be mutated.

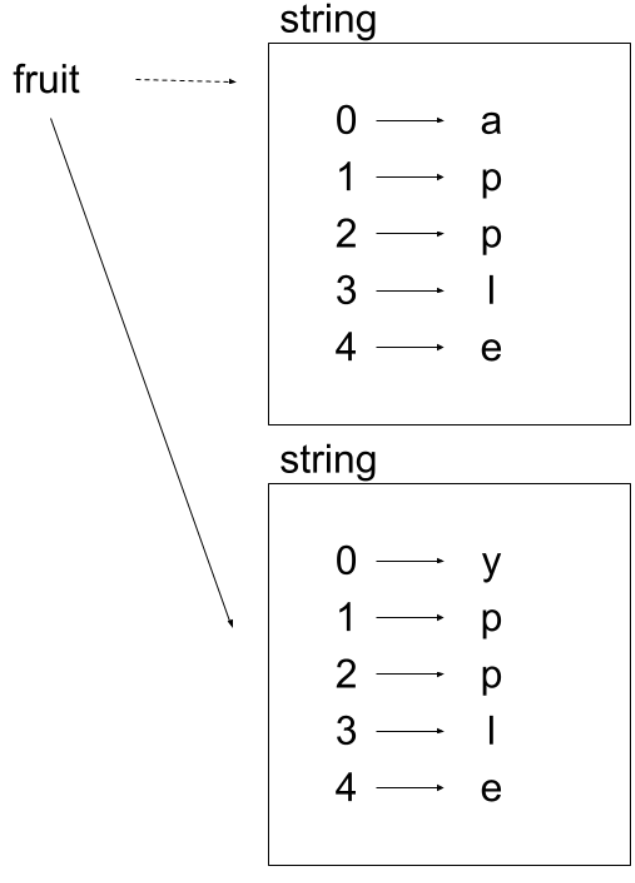


So, what are we to do to get a string with the value `'ypple'`? We simply have to create a new one.

In [8]:

```
fruit = 'apple'
print(fruit)
fruit = 'y' + fruit[1:] # Slice makes a copy
print(fruit)
```

```
apple
ypple
```


string
fruit
0 a
1 p
2 p
3 l
4 e
string
0 y
1 p
2 p
3 l
4 e

But... why?

- Performance: You know (most of) the storage requirements at construction time (during interpretation)

- Performance: You can safely reuse string objects (because you know they won't change!)

In [9]:

```
# Sharing is caring
my_fruit = 'apple'
your_fruit = 'apple'
print(my_fruit is your_fruit)
```

```
True
```

- Philosophy: "The other is that strings in Python are considered as "elemental" as numbers. No amount of activity will change the value 8 to anything else, and in Python, no amount of activity will change the string “eight” to anything else."
- History: Because the creator of Python (Guido van Rossum) decided it so

Getting the length of a string

When do you need the length of a string?

Password validation!

Username:

Password (8 characters minimum):

Sign in

How to get the length of a string

Use the `len` function.

In [10]:

```
password = input('Enter a password: ')
if len(password) < 8:
    print('Password must be at least 8 characters long.')
else:
    print('Password is valid!')
```

```
Enter a password: 12345678
Password is valid!
```

Traversing a string

When do you want to traverse a string?

Search engines

- Searching problem: Does a website's content contain the query? Return result
- Counting problem: How many times does a website's content contain the query? Prioritize result

How to traverse a string

1. `while` loop
2. `for` loop

`while` loop

In [13]:

```
query = 'p'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''

index = 0
while index < len(content):
    print(content[index])
    if content[index] == query:
        print('Found a match!')
        break # Why is breaking efficient?
    index += 1
```

```
A


p
Found a match!
```

In [14]:

```
query = 'puppy'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''

index = 0
while index < len(content) - len(query) + 1:
    word = content[index:index + len(query)]
    print(word)
    if word == query:
        print('Found a match!')
        break
    index += 1
```

```
A pup
 pupp
puppy
Found a match!
```

`for` loop

In [15]:

```
query = 'puppy'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''

for index in range(len(content) - len(query) + 1):
    word = content[index:index + len(query)]
    print(word)
    if word == query:
        print('Found a match!')
        break
```

```
A pup
 pupp
puppy
Found a match!
```

This pattern of traversing a string and returning (or breaking!) when we find what we are looking for is called a **search**.

The pattern of traversing a string and counting how many times we find what we are looking for is called a **count**.

In [16]:

```
query = 'puppy'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''

count = 0 # Initialize counter
for index in range(len(content) - len(query) + 1):
    word = content[index:index + len(query)]
    if word == query:
        print('Found a match!')
        count += 1 # Increment counter
print(f'Number of matches: {count}')
```

```
Found a match!
Found a match!
Found a match!
Number of matches: 3
```

String methods

What's the difference between methods and functions?

A method *is* a function!

```
fruits = ['apple', 'banana', 'canteloupe']
```

- Called by naming it and following the name with parentheses: `fruits.append('peach')`
- Can take input (arguments) and give output (return value): `fruits.pop() # Deletes and returns last element`

Specifically, methods are functions that *belong* to objects.

Which methods an object has depends on its type.

`fruit` has methods `append` and `pop` because it is a `list`.

A method is applied to the object it is called **on**.

You call a method **on** an object with dot notation: `<object>.<method>`.

`fruits.append('peach')` applies the `append` method on the `fruits` object, which is a list.

`upper` method

In [17]:

```
fruit = 'banana'
louder_fruit = fruit.upper()
print(louder_fruit)
```

```
BANANA
```

The `upper` method is called **on** the `fruit` object, which is a string. It returns the string with all characters in uppercase.

We know that `fruit` has a `upper` method because it is a string.

`find` method

In [18]:

```
fruit = 'banana'
index = fruit.find('a')
print(index)
```

```
1
```

The `find` method is called **on** the `fruit` object, which is a string. It searches for the substring (`'a'`) within the string and returns the index of the first match.

We know that `fruit` has a `find` method because it is a string.

You can find the complete list of string methods in the Python documentation: http://docs.python.org/3/library/stdtypes.html#string-methods.

The `in` operator

Just as with lists, the `in` operator takes two strings and returns `True` if the first is a substring of the second.

In [19]:

```
print('a' in 'banana')
```

```
True
```

In [20]:

```
query = 'puppy'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''
print(query in content)
```

```
True
```

So, why go through all the trouble of traversing a string?

In [21]:

```
query = 'puppy'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''

print(f'Using in operator: {query in content}')

for index in range(len(content) - len(query) + 1):
    word = content[index:index + len(query)]
    if word == query:
        print('Using for loop: True')
        break
    index += 1
```

```
Using in operator: True
Using for loop: True
```

What if you want to require two matches?

Cannot use `in` operator! Traversal to the rescue!

In [22]:

```
query = 'puppy'
content = '''A puppy is a juvenile dog.
A puppy's coat color may change as the puppy grows older,
as is commonly seen in breeds such as the Yorkshire Terrier.'''

count = 0
for index in range(len(content) - len(query) + 1):
    word = content[index:index + len(query)]
    if word == query:
        count += 1
        if count == 2:
            print('Found two matches!')
            break
    index += 1
```

```
Found two matches!
```

String comparison

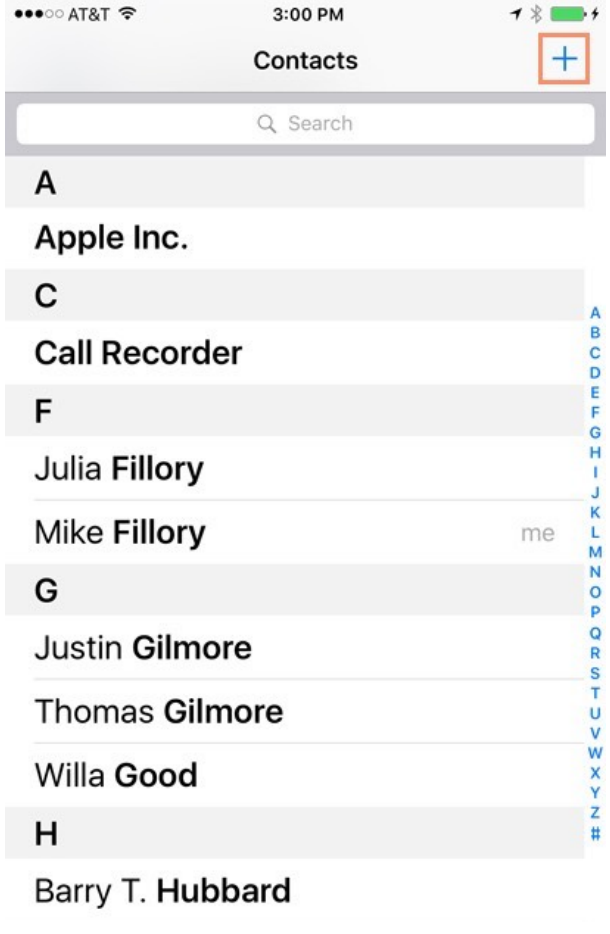
Relational operators (e.g. `==`, `<`, `>`) work on strings as well!

In [23]:

```
ans = input('Answer "yes" or "no": ')
if ans == 'yes':
    print('Answered "yes"!')
elif ans == 'no':
    print('Answered "no"!')
```

```
Answer "yes" or "no": yes
Answered "yes"!
```

Sorting strings



In [24]:

```
first_name = input('Enter first name: ')
last_name = input('Enter last name: ')
name = ' '.join([first_name, last_name])
contact_first_name = 'Julia'
contact_last_name = 'Fillory'
contact_name = ' '.join([contact_first_name, contact_last_name])

if last_name < contact_last_name:
    print(f'{name} comes before {contact_name}')
elif last_name > contact_last_name:
    print(f'{name} comes after {contact_name}')
else:
    if first_name < contact_first_name:
        print(f'{name} comes before {contact_name}')
    elif first_name > contact_first_name:
        print(f'{name} comes after {contact_name}')
```

```
Enter first name: Maria
Enter last name: Heinert
Maria Heinert comes after Julia Fillory
```

Be aware of casing when sorting!

Persistence

So far, the programs we've created are *transient* in nature--if we enter any data while they are running, that data is lost as soon as the programs end! When they are run again, they start with a clean slate.

- While, While West: Bank balance
- Love It or List It: Todo list

For more complex programs, we often want to save, or **persist**, the data that is generated during a program's run.

How would persistence improve the WWW game and the LIOLI todo list program?

One of the simplest ways to implement persistence is to save data in text files!

To save data in text files, you need to be able to:

- Read text files to get the data that is already saved
- Write to text files to save new data

Reading text files

Two steps to reading text files:

1. Create a file object
2. Use that file object to read text file

How to create a file object

Use the `open` function. It takes as its first argument the file to open.

In [25]:

```
fin = open('words.txt') # fin = "file in"
```

Doesn't do much!

How to use a file object to read text files: `readline`

To `fin.read` or to `fin.readline`, that is the question...

`read`

Will return *all* of the text in the file at once!

This is okay when the file is relatively small...

In [26]:

```
fin = open('hamlet.txt')
fin.read()
fin.close() # Good practice!
```

`readline`

But, when you have a large file, like `words.txt`,

```
1 aa
2 aah
...
113783 zymurgy
```

it usually makes more sense to read the file line-by-line.

```
fin = open('words.txt')
text = fin.read() # Will require ~1 MB of memory
```

In [27]:

```
fin = open('words.txt')
print(fin.readline())
print(fin.readline()) # What's with the weird spacing?
fin.close()
```

```
aa

aah

```

You can use the string method `strip` to *strip* away whitespace from the beginning and end of a string.

In [28]:

```
fin = open('words.txt')
print(fin.readline().strip())
print(fin.readline().strip())
fin.close()
```

```
aa
aah
```

So, `readline` will read a text file line-by-line...

Is there any way to read all of `words.txt` line-by-line *without* having to call `readline` 113,783 times?

How to use a file object to read text files: `for` loop

Use a `for` loop to iterate through the file object!

```
fin = open('words.txt')
for line in fin:
    word = line.strip()
    print(word)
fin.close()
```

In [29]:

```
fin = open('hamlet.txt')
print(fin.read())
fin.close()
```

```
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
```

In [30]:

```
fin = open('hamlet.txt')
for line in fin:
    line = line.strip()
    print(line)
fin.close()
```

```
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
```

In [31]:

```
fin = open('hamlet.txt')
print(fin.read())

for line in fin:
    line = line.strip()
    print(line)

fin.close()
# Why isn't the text file printed twice?
```

```
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
```

In [32]:

```
fin = open('hamlet.txt')
print(fin.read())

fin.seek(0) # Set the file object's position to the 0th byte

for line in fin:
    line = line.strip()
    print(line)

fin.close()
```

```
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
```

Writing to files

Two steps to writing to text files:

1. Create a file object with the 'write', `w`, option
2. Use that file object to write text to the file

How to create a file object with the write option

Use the `open` function. Pass in the file to open as the first argument and `'w'` as its second.

In [ ]:

```
fout = open('todo-list.txt', 'w') # fout = "file out"
```

Doesn't do much!

How to use a file object to write to text files

Use the `write` function, which takes the text to write to the file.

In [33]:

```
fout = open('todo-list.txt', 'w')
fout.write('Paint walls\n')
fout.write('Install trim\n')
fout.write('Replace light fixture\n')
fout.close()

fin = open('todo-list.txt')
print(fin.read())
fin.close()
```

```
Paint walls
Install trim
Replace light fixture
```

What could possibly go wrong when working with text files?

Exception = Something **exception**ally bad happened

```
fin = open('does-not-exist.txt')
```

File doesn't exist!

```
fin = open('not-your-file.txt')
```

You don't have permission to access a file!

```
fin = open('deliverables/src')
```

Is a directory!

How do we avoid exceptions?

We *could* try to check for anything that could go wrong:

```
import os
file = 'file.txt'
if os.path.isfile(file) and os.access(file, os.R_OK):
    fin = open(file)
```

Or, we could just **try** the risky code and handle any exceptions that arise.

In [34]:

```
try:
    fin = open('file.txt') # Risky business
except:
    print('Something went wrong!')
```

```
Something went wrong!
```

Because it is difficult to account for all the possible ways using a file can go wrong, it is preferable to just **catch exceptions** as they occur.

Summary

- What strings are used for and how to create them
- How to access character(s) in a string
- Why strings are immutable
- How to get the length of a string
- How to traverse a string with `while` and `for` loops
- How to use string methods
- How to use the `in` operator
- How to compare strings

- What persistence is
- How to read and write to text files
- How to catch exceptions when working with text files

Office Hours

M 11:30 AM - 12:30 PM; 2:00 - 4:00 PM

Tu 10:00 AM - 1:00 PM

W 11:30 AM - 12:30 PM

F 11:30 AM - 12:30 PM; 3:00 - 4:00 PM