The Spam Van

At the end of this course, we want to create a basic sales system for our circus food truck — the Spam Van.

... But first, we will start with **a list of British slang** to *inspire* our ***menu.***

FLYING THROUGH PYTHON

Level 1

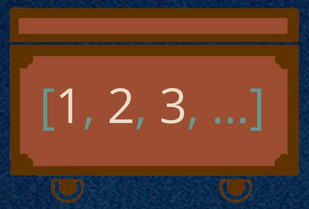
Slanguage

Section 1 – Lists

Flying through Python

A List Is a Container of Things

Lists in Python are containers that can store anything you want.




```
# an empty list
empty = []
```

```
# list of numbers
nums = [10, 20, 30, 40.5]
```

```
# list of strings
words = ['cheerio', 'cheers', 'watcha', 'hiya']
```

```
# list of mixed items
anything = [10, 'hello', 'ahoy', 123.45]
```

Note: A Python list is called an **array** in other languages, like JavaScript.

FLYING THROUGH PYTHON

An Item's Index Is Its Position

List items are ordered starting with index (or position) 0.

```
greetings = ['cheers', 'cheerio', 'watcha', 'hiya']
```

index → [0] [1] [2] [3]

```
>>> greetings[0]
```
→ `'cheers'`

```
>>> greetings[3]
```
→ `'hiya'`

```
>>> greetings[1:3]
```
→ `['cheerio', 'watcha']`

You can also slice a list to get a sub-section of the list.

FLYING THROUGH PYTHON

Creating a List and Adding Items

Let's create a list of British sayings.

```
slang = ['cheerio', 'pip pip', 'smashing']
print(slang)
```

Can create the list with elements in it already, or empty

```
british = []
```

```
['cheerio', 'pip pip', 'smashing']
```

We can add items as we need.

```
slang.append('cheeky')
slang.append('yonks')
print(slang)
```

append() adds new items to the end of our list.

list_name.append(item)

```
['cheerio', 'pip pip', 'smashing', 'cheeky', 'yonks']
```

FLYING THROUGH PYTHON

Removing Elements

We can remove items from our list by using the value or the index.

```
slang = ['cheerio', 'pip pip', 'smashing', 'cheeky', 'yonks']
```

[0] [1] [2] [3] [4]

```
slang.remove('cheeky')
```

Either OR
depending on whether
you know the value or the index

```
del slang[3]
```

```
print(slang)
```

```
['cheerio', 'pip pip', 'smashing', 'yonks']
```

We can also delete a slice.

```
slang = ['cheerio', 'pip pip', 'smashing', 'yonks']
del slang[:2]
print(slang)
```

```
['smashing', 'yonks']
```

FLYING THROUGH PYTHON

Level 1

Slanguage

Section 2 – Dictionaries

Flying through Python

Maintaining Two Lists

What if we also wanted a list to keep track of the American definitions of our British slang list?
If we do anything to one of our lists, we have to duplicate the operation in the other list.

```
slang       = ['cheerio', 'pip pip', 'smashing', 'yonks', 'knackered']
translation = ['goodbye', 'goodbye', 'terrific', 'ages',  'tired']
                 [0]        [1]         [2]        [3]       [4]

del slang[0]
del translation[0]

print(slang)
print(translation)
```

If we add or delete from one list...
we have to do the same thing in the other list.

```
['pip pip', 'smashing', 'yonks', 'knackered']
['goodbye', 'terrific', 'ages',  'tired']
```

We can improve upon this with dictionaries...

FLYING THROUGH PYTHON

A Dictionary Maps Keys to Values

A dictionary is similar to a list, but you look up values by using a key instead of an index.

key	value
'cheerio'	'goodbye'
'knackered'	'tired'

Great for our slang translator —
we can look up the definition of a word with the word!

```
slang = {'cheerio':'goodbye', 'knackered':'tired', 'yonks':'ages'}
```

Key Assign to Value

Each item is known as
a "key-value pair"

```
print(slang['cheerio'])
```

To look up a **value** in a dictionary,
we send in a **key**
(instead of index in a list).

→ 'goodbye'

FLYING THROUGH PYTHON

Dictionaries Can Hold Anything

Dictionaries, like lists, can hold anything you want: numbers, strings, a mix of both, and other objects.

Dictionary of strings to strings

```
slang = {'cheerio':'goodbye', 'rubbish':'trash'}
```

Dictionary of strings to numbers

```
menu = {'spam meal':15, 'spam with fries':10}
```

A menu item's name is the key, and its price is the value.

Dictionary of anything

```
anything = {10:'hello', 2:123.45}
```

FLYING THROUGH PYTHON

Creating a Dictionary and Adding Values

Create an empty dictionary:

```
slang = {}
```

Adding dictionary items:

```
slang['cheerio']   = 'goodbye'
slang['smashing']  = 'terrific'
slang['knackered'] = 'tired'
print(slang)
```

This is how you add key-value pairs, and also how you can update values.

```
{'cheerio': 'goodbye', 'smashing': 'terrific', 'knackered': 'tired'}
```

Updating Values in Our Dictionary

Our current dictionary:

```
print(slang)
```

→ `{'cheerio': 'goodbye', 'smashing': 'terrific', 'knackered': 'tired'}`

Let's update the value for smashing...

```
slang['smashing'] = 'awesome'
```

... and look up and print the same value.

```
print(slang['smashing'])
```

→ `awesome`

Removing Dictionary Items

You can delete an item from a dictionary by looking up its key.

```
slang = {'cheerio':'goodbye', 'knackered':'tired', 'smashing':'terrific'}

del slang['cheerio']

print(slang)
```

del will delete the key-value pair from the dictionary

```
{'smashing': 'terrific', 'knackered': 'tired'}
```

FLYING THROUGH PYTHON

Get an Item That Might Not Be There

Trying to access a key that doesn't exist will cause an error.

```
slang['bloody']
```

```
KeyError: 'bloody'
```

If you want to check if a word is in the dictionary, use get().

```
# Get a value that might not be there
result = slang.get('bloody')
print(result)
```

Using get() is a best practice because it won't crash your program.

If the word WAS NOT in the dictionary

```
None
```

None is a type that represents the absence of a value.

FLYING THROUGH PYTHON

None Type

None represents the absence of a value, and also evaluates to False in a conditional.

Result equals None because the key doesn't exist

If exists...

Otherwise

```
slang = {'cheerio':'goodbye', 'knackered':'tired'}
result = slang.get('bloody')

if result:
    print(result)
else:
    print('Key doesn't exist')
```

```
Key doesn't exist
```

FLYING THROUGH PYTHON

Using a Dictionary to Translate a Sentence

Let's translate each slang word in a sentence:

```
slang = {'mate':'buddy', 'knackered':'tired', 'cheeky':'offensive'}

sentence =    'Sorry, my ' + 'mate' + "'s a bit " + 'cheeky' + '!'

translation = 'Sorry, my ' + slang.get('mate') + "'s a bit " +
               slang.get('cheeky') + '!'

print(sentence)
print(translation)
```

```
Sorry, my mate's a bit cheeky!
Sorry, my buddy's a bit offensive!
```

FLYING THROUGH PYTHON

Level 1

Slanguage

Section 3 – Comparing and Combining Lists and Dictionaries

Flying through Python

Comparing Lists

How do we check if two ***lists*** are ***equal***?

```
my_list =   [1, 2, 3, 4]
your_list = [4, 3, 2, 1]
his_list =  [1, 2, 3, 4]

print(my_list == your_list)

print(my_list == his_list)
```

```
False
```

```
True
```

Two lists need to have the **same values** in the **same order** to be equal.

FLYING THROUGH PYTHON

Comparing Dictionaries

How do we check if two ***dictionaries*** are ***equal***?

```
my_dict =   {1:1, 2:2, 3:3, 4:4}
your_dict = {4:4, 3:3, 2:2, 1:1}

print(my_dict == your_dict)
```

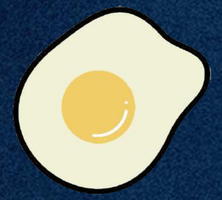
→ `True`

Since dictionaries aren't ordered, they only need to have the **same key-value pairs** to be equal.

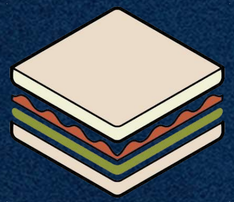
FLYING THROUGH PYTHON

Breakfast, Lunch, and Dinner Menus

Let's say we have three separate menu lists: Breakfast, Lunch, Dinner...
How would we combine these into one list?



```
breakfast = ['Spam n Eggs', 'Spam n Jam', 'Spam n Ham']
```



```
lunch     = ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)']
```

```
dinner    = ['Spalad', 'Spamghetti', 'Spam noodle soup']
```

One list?

FLYING THROUGH PYTHON

A *List* of *Lists*

You can have a container of containers — Menus is a list of menus, and each menu is a list of foods.

list of lists

```
menus = [ ['Spam n Eggs', 'Spam n Jam', 'Spam n Ham'],
          ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)'],
          ['Spalad', 'Spamghetti', 'Spam noodle soup']    ]

print('Breakfast Menu:\t', menus[0])
print('Lunch Menu:\t',     menus[1])
print('Dinner Menu:\t',    menus[2])
```

```
Breakfast Menu: ['Spam n Eggs', 'Spam n Jam', 'Spam n Ham']
Lunch Menu:     ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)']
Dinner Menu:    ['Spalad', 'Spamghetti', 'Spam noodle soup']
```

Getting an Item From a Two-dimensional List

You can use two indexes to get an individual item.

How would you get this item?

```
menus = [['Spam n Eggs', 'Spam n Jam', 'Spam n Ham'],
         ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)'],
         ['Spalad', 'Spamghetti', 'Spam noodle soup']]

print(menus[0])
```

The list at index 0 in the outer list

```
['Spam n Eggs', 'Spam n Jam', 'Spam n Ham']
```

A good start! Now we just need to get the second item in this list.

Getting an Item From a Two-dimensional List

You can use two indexes to get an individual item.

How would you get this item?

```
menus = [ ['Spam n Eggs', 'Spam n Jam', 'Spam n Ham'],
          ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)'],
          ['Spalad', 'Spamghetti', 'Spam noodle soup']]

print(menus[0][1])
```

The list at index 0 in the outer list,
the item at index 1 in the inner list

```
'Spam n Jam'
```

It would be better if we didn't have to know breakfast was at index 0, lunch at 1, etc.

A *Dictionary* of *Lists*

We can use a dictionary with keys for Breakfast, Lunch, and Dinner.

Dictionary where... each value is a list

```
menus = {'Breakfast': ['Spam n Eggs', 'Spam n Jam', 'Spam n Ham'],
         'Lunch': ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)'],
         'Dinner':  ['Spalad', 'Spamghetti', 'Spam noodle soup'] }

print('Breakfast Menu:\t', menus['Breakfast'])
print('Lunch Menu:\t',     menus['Lunch'])
print('Dinner Menu:\t',    menus['Dinner'])
```

```
Breakfast Menu: ['Spam n Eggs', 'Spam n Jam', 'Spam n Ham']
Lunch Menu:     ['SLT (Spam-Lettuce-Tomato)', 'PB&S (PB&Spam)']
Dinner Menu:    ['Spalad', 'Spamghetti', 'Spam noodle soup']
```

Level 2

Loopty Loops

Section 1 – For Loops and range()

Flying through Python

The Spam Van

At the end of this course, we want to create a basic sales system for our circus food truck — the Spam Van.

Let's calculate the average price of our menu so that we can prove we deserve a single **$**$$$ rating on sites like Yelp.

To do this, we'll use a loop to look at each price in our price list.

FLYING THROUGH PYTHON

for Loop: For Each item in Our List, Do This

```
prices = [2.50, 3.50, 4.50]

for price in prices:
    print('Price is', price)
```

price is a temporary variable that holds one of the **prices** in the list for each run.

Like saying "do this" for each number **price** in our list **prices**

The loop runs one time for each price in the list.

	price		Loop Output
1st loop	2.50	→	Price is 2.5
2nd loop	3.50	→	Price is 3.5
3rd loop	4.50	→	Price is 4.5

FLYING THROUGH PYTHON

Using a for Loop to Sum Our Prices

Let's look at our for loop for summing the prices in action.

We can add to our total and print total to see it changing.

```
# Average price
total = 0
prices = [2.50, 3.50, 4.50]

for price in prices:
    print('Price is', price)
    total = total + price
    print('total is', total)
```

```
price is 2.5
total is 2.5
price is 3.5
total is 6
price is 4.5
total is 10.5
```

total = 0 + 2.5

total = 2.5 + 3.5

total = 6 + 4.5

Averaging the Prices After the Loop Ends

```
# Average price
total = 0
prices = [2.50, 3.50, 4.50]

for price in prices:
    total = total + price

average = total/len(prices)
print('avg is', average)
```

After the loop,
we compute the average.

The print statements were just so that we could "see" what was happening, so we can remove them now.

```
avg is 3.5
```

Generating Raffle Tickets

We also want to do something fun for our customers — with each purchase, they get a raffle ticket, and then we draw 10 random ticket numbers at the end of the day and give out prizes.

We'll need to:

(1) **Generate a random ticket number**

(2) **Repeat that 10 times — we can use a loop to do this!**

FLYING THROUGH PYTHON

Generating a Random Ticket Number

First, how do we get a **single** random number?

```
import random

r1 = random.random()
print(r1)
```

```
0.80945
```

Gives us a random number from [0.0, 1.0)

```
r2 = random.choice([1,2,3,4,5])
print(r2)
```

```
3
```

Gives us a random choice from a list

```
r3 = random.randint(1, 1000)
print(r3)
```

```
902
```

Gives us a random number in this range

This will allow us to pick a random raffle ticket from 1 to 1,000

Now we just need to do that 10 times...

FLYING THROUGH PYTHON

Generating 10 Random Raffle Tickets

```
import random

for i in ???:
    ticket = random.randint(1, 1000)
    print(ticket)
```

What do we put here if we don't have a list?

How do we iterate through the for loop 10 times?

```
range(10)
```

range(count) generates a list of [0, ..., count-1] behind the scenes.

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

FLYING THROUGH PYTHON

Using range() in Our for Loop

```python
import random

for i in range(10):
    ticket = random.randint(1, 1000)
    print(ticket)
```

```
698
689
328
113
707
958
440
297
189
899
```

We don't have to use `i` in our loop,
but we can if we want to...

FLYING THROUGH PYTHON

Getting More Specific With range()

We've held our circus in Orlando, Florida, every other year since 2005 until now — 2016. How would we list the years?

start stop step

```
for i in range(2005, 2016, 2):
    print(i)
```

We can also call range() with more parameters: start, stop, and step.

```
2005
2007
2009
2011
2013
2015
```

Add 2

Stop before 2016

FLYING THROUGH PYTHON

Level 2

Loopty Loops

Section 2 – The Spam Van Menu

Flying Through Python

The Spam Van

At the end of this course, we want to create a basic sales system for our circus food truck — the Spam Van.

... Let's start creating the menu with **for loops**

FLYING THROUGH PYTHON

Create Our Monty Python Restaurant Menu

For each British slang word, let's create a menu item made with Spam.

```
slang = ['Knackered', 'Pip pip', 'Squidgy', 'Smashing']
```

```
menu = []

for word in slang:
    menu.append(word + ' Spam')

print(menu)
```

For each slang word in our list, we'll concatenate Spam to it, and we'll add that to our menu list.

```
['Knackered Spam', 'Pip pip Spam', 'Squidgy Spam', 'Smashing Spam']
```

Assigning Prices to Our Menu Items

For each of our menu items, let's assign prices and use a dictionary to store them.

```
menu = ['Knackered Spam', 'Pip pip Spam', 'Squidgy Spam', 'Smashing Spam']
```

$0.50 $1.50 $2.50 $3.50

```
# Let's store the menu items with their prices in a dictionary
menu_prices = {}
price = 0.50
for item in menu:
    menu_prices[item] = price
    price = price + 1

print(menu_prices)
```

The item is the key — for example, 'Knackered Spam' is the key and $0.50 is the value.

```
{'Pip pip Spam' : 1.50, 'Squidgy Spam': 2.50,
 'Smashing Spam': 3.50, 'Knackered Spam': 0.50}
```

FLYING THROUGH PYTHON

Printing Our Menu Items

Now the following dictionary we created is stored in memory.

```
menu_prices = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50,
               'Squidgy Spam': 2.50, 'Smashing Spam': 3.50}
```

Could we print out each item in our dictionary in a loop like a list?

```
for item in menu_prices:
    print(item)
```

This will default to just printing the keys in a dictionary.

```
Pip pip Spam
Squidgy Spam
Smashing Spam
Knackered Spam
```

FLYING THROUGH PYTHON

Using a Dictionary's Key and Value in a for Loop

```
menu_prices = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50,
               'Squidgy Spam': 2.50, 'Smashing Spam': 3.50}

for name, price in menu_prices.items():
    print(name, ': $', price)
```

Now the loop has access to both the key and the value here — name and price.

```
Pip pip Spam : $ 0.5
Squidgy Spam : $ 1.5
Smashing Spam : $ 2.5
Knackered Spam : $ 3.5
```

We'd like to format this better, though, by getting rid of these spaces...

Note: To get a list of the keys in a dictionary, use `dict_name.keys()`
and to get a list of the values in a dictionary, use `dict_name.values()`

FLYING THROUGH PYTHON

Formatting Our Menu Items

Let's get rid of the unnecessary spaces in our output.

```
menu_prices = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50,
               'Squidgy Spam': 2.50, 'Smashing Spam': 3.50}

for name, price in menu_prices.items():
    print(name, ': $', price, sep='')
```

print() defaults to having a space to separate each word. We can override this by setting sep = empty string.

```
Pip pip Spam: $0.5
Squidgy Spam: $1.5
Smashing Spam: $2.5
Knackered Spam: $3.5
```

Looking better — now we just need to add rounding to two decimal places.

FLYING THROUGH PYTHON

Rounding a Float to a Certain Number of Decimal Places

Since $1.5 doesn't look as nice as $1.50, we'll need to round to two decimal places.

```
menu_prices = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50,
               'Squidgy Spam': 2.50, 'Smashing Spam': 3.50}

for name, price in menu_prices.items():
    print(name, ': $', format(price, '.2f'), sep='')
```

2 decimal places,
f for float format

We can format our price float to two decimal places using the built-in format() function.

```
Pip pip Spam: $0.50
Squidgy Spam: $1.50
Smashing Spam: $2.50
Knackered Spam: $3.50
```

Looks great!

FLYING THROUGH PYTHON

Level 2

Loopty Loops

Section 3 – While Loops

Flying Through Python

Interface for Ordering Menu Items

We want to let customers order and add as many menu items as they want. A for loop wouldn't be good for this because we don't know how many times to loop...

Python 3.4.3 Shell

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What would you like to order? (Q to Quit)
```

FLYING THROUGH PYTHON

A Simple while Loop Example

A while loop would be better since **while** loops continue looping ***while*** a condition is **True.**

```
x = 1

while  x != 3:
    print('x is', x)
    x = x + 1
```

Like saying "do this" as long as x does not equal 3

The loop would run forever unless we changed x so the conditional could eventually be false.

	x	Loop Output
1st loop	1	x is 1
2nd loop	2	x is 2
3rd loop	3	loop is done running

FLYING THROUGH PYTHON

Let’s Create Our Interface for Ordering From a Menu

We’ll use a while loop to let customers order and add as many menu items as they want.

```
orders = []
order = input("What would you like to order? (Q to Quit)")

while (order.upper() != 'Q'):
    # Find the order and add it to the list if it exists
    # See if the customer wants to order anything else
```

Adding Menu Items to the Orders List

We only want to add the customer's order if it exists on the menu.

I left the order of the red arrows and code in the green box because the instructor had it set up that way. Let me know if it needs to be different!

```
orders = []
order = input("What would you like to order? (Q to Quit)")

while (order.upper() != 'Q'):
    # Find the order and add it to the list if it exists

    found = menu.get(order)
    if found:
        orders.append(order)
    else:
        print("Menu item doesn't exist")

    # See if the customer wants to order anything else
```

Try to get() the menu item

If it exists, add it to the list

If not, let the customer know

If we don't change "order" in the loop, it will never be "Q" and the loop will never stop running.

FLYING THROUGH PYTHON

Asking the Customer if They Want to Order More

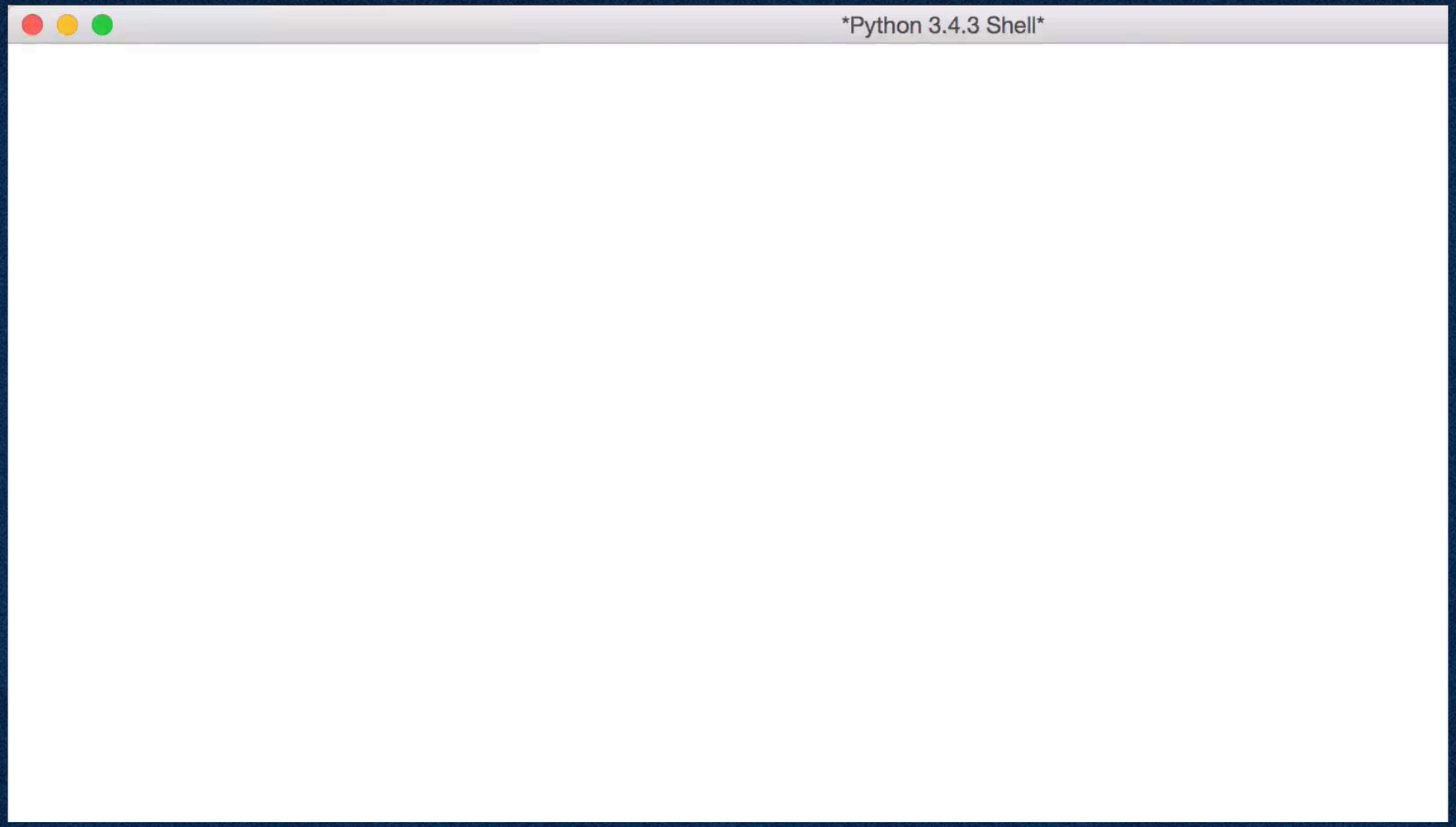
We want to let customers order and add as many menu items as they want.

```
orders = []
order = input("What would you like to order? (Q to Quit)")

while (order.upper() != 'Q'):
    # Find the order and add it to the list if it exists
    found = menu.get(order)
    if found:
        orders.append(order)
    else:
        print("Menu item doesn't exist")


    # See if the customer wants to order anything else
    order = input("Anything else? (Q to Quit)")

print(orders)
```

Wasn't letting me put print (orders) with the rest of the code above it.

YING ROUGH PYTHON

Ordering Interface in Action



FLYING THROUGH PYTHON

Another Way to Break Out of the Loop

A break statement will exit the loop immediately and continue running the program.

```
orders = []
order = input("What would you like to order? (Q to Quit)")

while (order.upper() != 'Q'):(True):

    if order.upper() == 'Q':
        break

    # Find the order and add it to the list if it exists
    …


    # See if the customer wants to order anything else
    order = input("Anything else? (Q to Quit)")

print(orders)
```

Always run the loop forever...

... unless the user types "Q," then break out of the loop.

FLYING THROUGH PYTHON

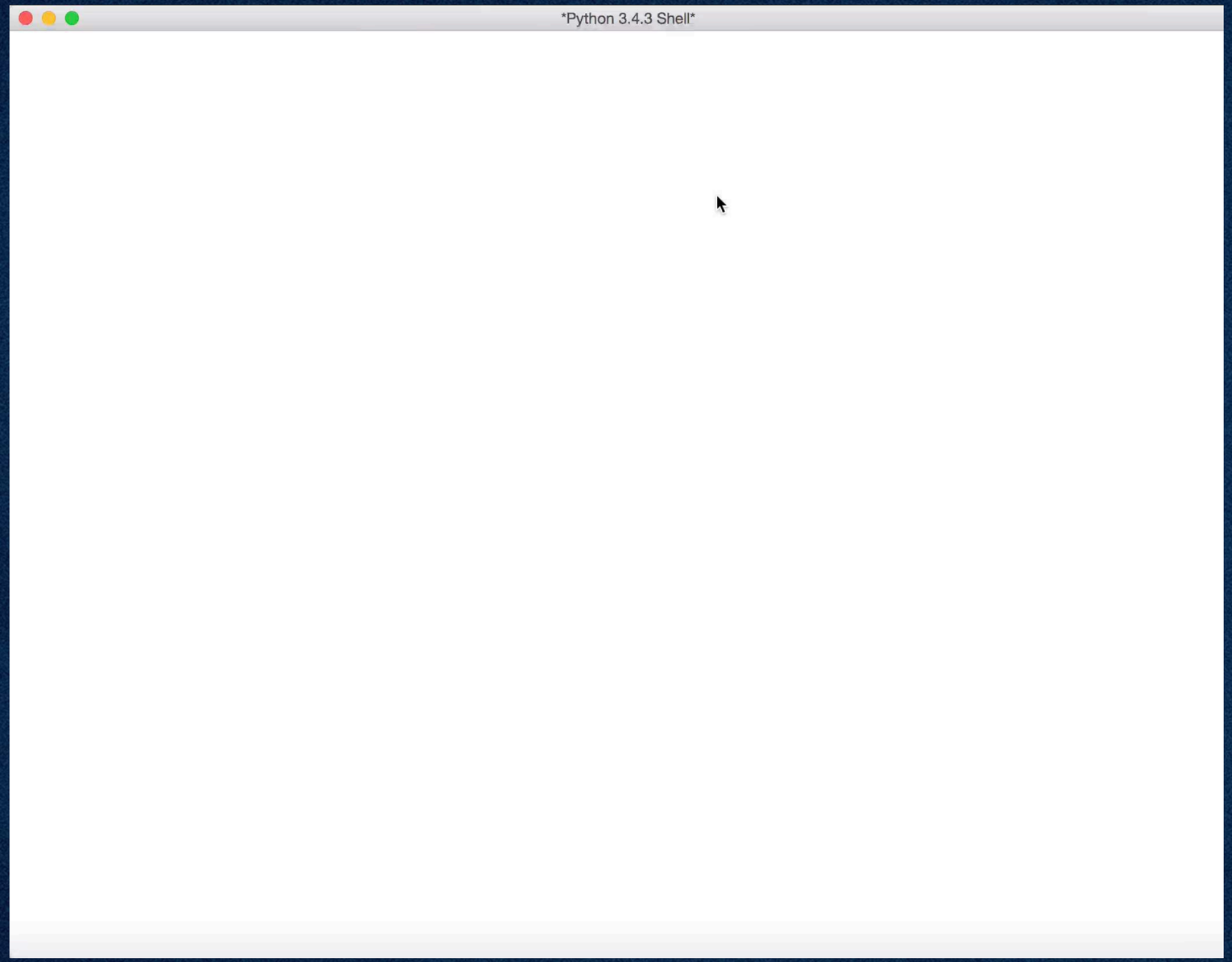
Using Continue for Special Cases in the Loop

We have a problem — we ran out of Cheeky Spam! We'll need to let customers know...

```
orders = []
order = input("What would you like to order? (Q to Quit)")

while (True):
    if order == 'Cheeky Spam':
        print('Sorry, we're all out of that!')
        continue
    if order.upper() == 'Q':
        break

    # Find the order and add it to the list if it exists
    …
    # See if the customer wants to order anything else
    order = input("Anything else? (Q to Quit)")

print(orders)
```

If the order is Cheeky Spam,
skip the rest of the code in the loop one time,
go to the top, and start running again.

Infinite Loops Are Bad



Oops, we accidentally created an infinite loop!

If this happens to you:
- Stay calm
- Press Ctrl+C

FLYING THROUGH PYTHON

Fixing our Infinite Loop

We need to get the customer's order at the top of our loop now so they can change their order.

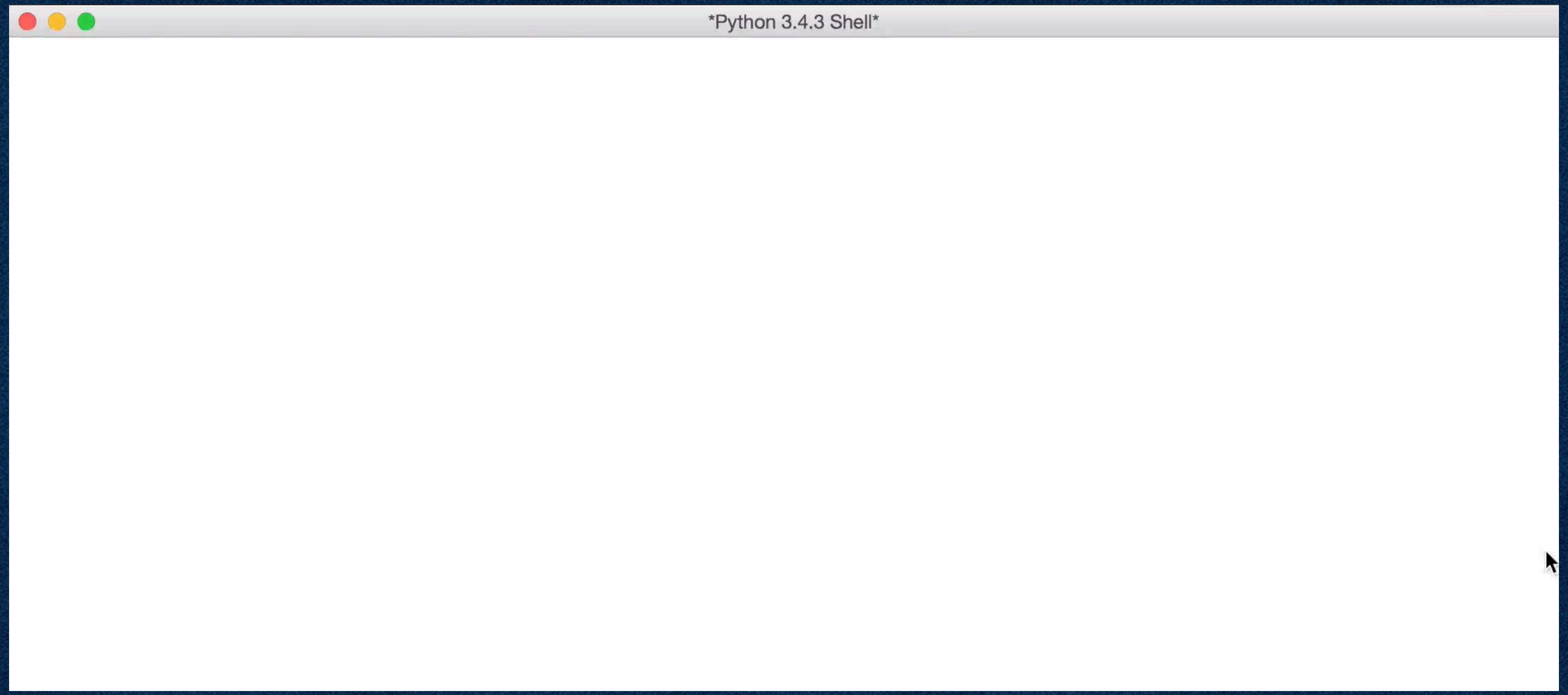
```
…
while (True):
    # See if the customer wants to order anything else
    order = input("Can I take your order? (Q to Quit)")

    if order == 'Cheeky Spam':
        print("Sorry, we're all out of that!")
        continue
    elif order.upper() == 'Q':
        break

    # Find the order and add it to the list if it exists
    …

print(orders)
```

Move the ordering to the top of the loop.

Now if they order Cheeky Spam, they can order again.

FLYING THROUGH PYTHON

Ordering Interface Demo



FLYING THROUGH PYTHON

Level 3

Functions in Flight

Section 1 – Introduction to Functions

Flying through Python

Problem: We're Repeating a Lot of Code

We've got this code that gets the average price

```
# Average price
total = 0
prices = [2.50, 3, 4.50, 5]

for price in prices:
    total = total + price

average = total/len(prices)
```

And this code that gets the average daily customers

```
# Average daily customers
total = 0
customers = [29, 21, 55, 5
             10, 14, 12]

for cust in customers:
    total = total + cust

average = total/len(customers)
```

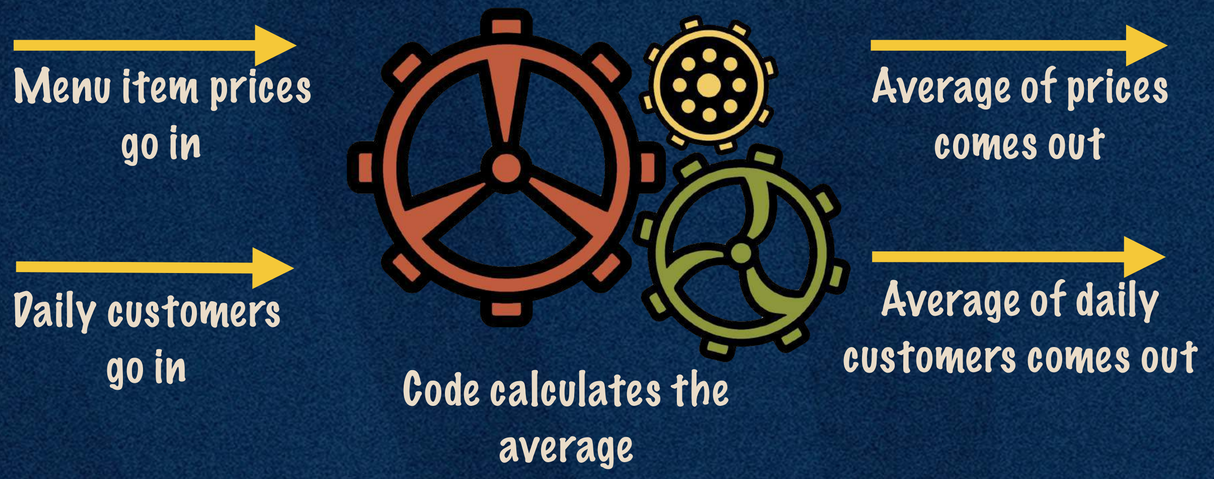
And we also want to write code to calculate average daily sales...

Wouldn't it be great to just have one piece of code that can average anything we ask it to?

FLYING THROUGH PYTHON

Functions Perform Specific Tasks

Functions are like mini programs that allow us to perform a specific task.



The code performing these calculations is a **function**.

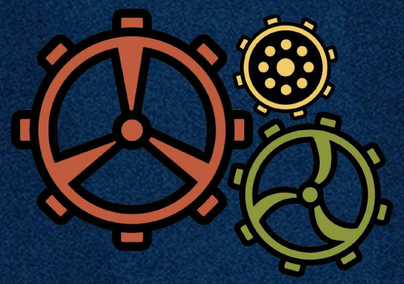
FLYING THROUGH PYTHON

Some Functions We've Used Before

We've been using functions all along, like print():

```
print('Welcome to the Spam Van!')
```

The String 'Welcome to the Spam Van!'



print()

Output to the console

Another function we've seen before, range():

```
my_list = range(10)
```

The number 10

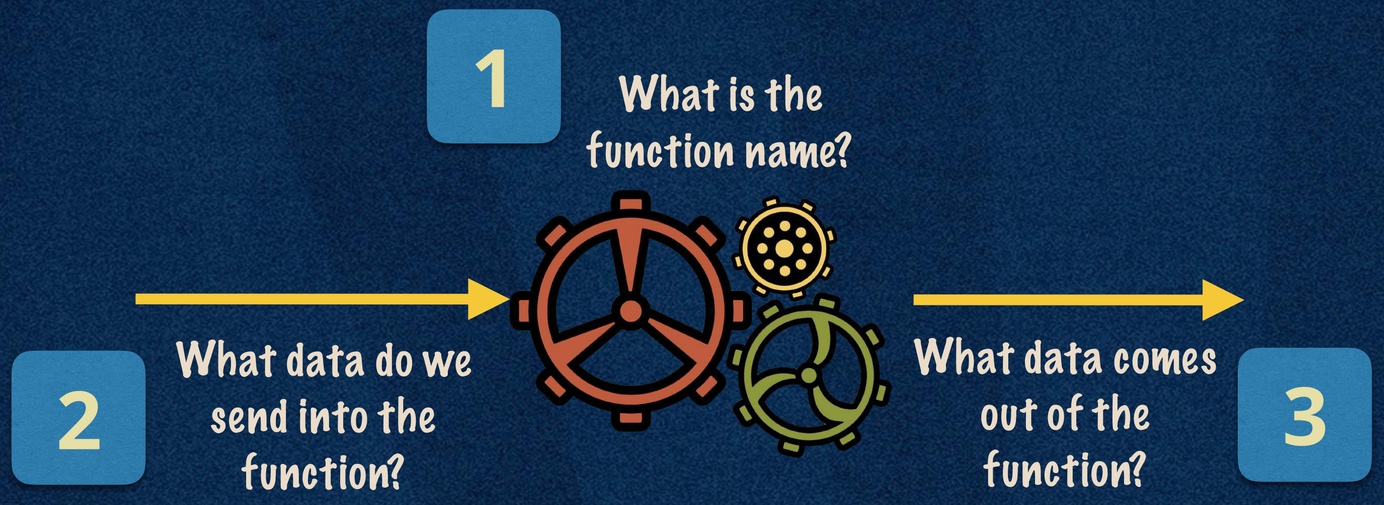
range()

Return a list from 0 to 9

FLYING THROUGH PYTHON

Making Our Own Function

We need to define three things to make our own function.



FLYING THROUGH PYTHON

Step One: Naming the Function

The name of the function here is “average.”

Name:
The name of the function is how we’ll run it later.

Every function starts with `def`.

```
def average():
    ...
```

Colon :
A colon marks the beginning of the function’s code.

FLYING THROUGH PYTHON

Step Two: Declaring Function Parameters

These parameters are variables that can be used in the function.

```
def average(prices):
    ...
```

Parameters:
Any variables we need the function to use

Flying through Python

Step Three: Return Data From the Function

```
def average(prices):
    ...

    return avg
```

Indented:
The code inside of the function is indented.

Return value:
Functions may or may not return values — this function returns the value of a variable called avg.

Returning data from a function is optional.

FLYING THROUGH PYTHON

Putting Code in the average() Function

We could take our average price code from Level 2 and make that its own function.

Our code from Level 2

```
# Average price
total = 0
prices = [29, 21, 55, 10]
for price in prices:
    total = total + price

avg = total/len(prices)
print(avg)
```

```
def average(prices):
    total = 0

    for price in prices:
        total = total + price

    avg = total/len(prices)
    return avg
```

Parameter

Indented

Return value

FLYING THROUGH PYTHON

Calling the average() Function

Functions don't run until you call them.

Assignment

```
numbers = [1,2,3,4,5]
my_average = average(numbers)
print(my_average)
```

Function call

Our parameter is a list of numbers.

```
3
```

my_average now contains the value 3 — the result of calling the average function

FLYING THROUGH PYTHON

Generalizing Our average() Function

We want to use our function for getting the average for other lists of numbers, not just prices.

Before

```
def average(prices):
    total = 0

    for price in prices:
        total = total + price

    avg = total/len(prices)
    return avg
```

After

```
def average(numbers):
    total = 0

    for num in numbers:
        total = total + num

    avg = total/len(numbers)
    return avg
```

Change the name of the parameter to be more general.

Notice how this general average function can average the numbers in any list!

For example, we can use it here...

```
daily_sales = [10,14,8]
result = average(daily_sales)
```

The variable name of our list doesn't have to match what it's called in the function.

FLYING THROUGH PYTHON

How Our Function Executes Code

This is our file with our function defined, followed by our main program where we use our function.

```
def average(numbers):
    total = 0
    for num in numbers:
        total = total + num

    avg = total/len(numbers)
    return avg

# Use our function on prices
prices = [2.50, 3, 4.50, 5]

result = average(prices)

print(result)
```

But inside the function, that data is accessed in a variable named numbers.

Program starts here:

The prices array is sent as an input to the function.

```
3.75
```

FLYING THROUGH PYTHON

Level 3

Functions in Flight

Section 2 – main()

Flying through Python

main() and Calling main()

A best practice is to organize our main program code into a function called... main().

Before

```
def average(numbers):
    total = 0
    for num in numbers:
        total = total + num

    avg = total/len(numbers)
    return avg


# main
prices = [29, 21, 55, 10]

result = average(prices)

print(result)
```

The main part of our program where all of the execution starts

Organize our main code into a main function.

Still need to call main() to execute our program.

After

```
def average(numbers):
    total = 0
    for num in list:
        total = total + num

    avg = total/len(numbers)
    return avg


def main():
    prices = [29, 21, 55, 10]

    result = average(prices)

    print(result)

main()
```

Order of Execution

Let's look at the order that each line of our program gets run in.

```
def average(numbers):
    total = 0
    for num in list:
        total = total + num

    avg = total/len(numbers)
    return avg

def main():
    prices = [29, 21, 55, 10]
    result = average(prices)

    print(result)

main()
```

* The average() function will only run when it gets called.

2 Then runs all of the code in main()

1 All of the execution starts here!

Since these are functions, they don't get run until they're called.

Understanding Local Scope

```
def average(numbers):
    total = 0
    for num in list:
        total = total + num

    avg = total/len(numbers)
    return avg

def main():
    prices = [29, 21, 55, 10]

    result = average(prices)
    print(total)
    print(result)

main()
```

Local scope

The variables declared in the average function only exist within the function.

Their scope is local to this function.

If we try to access total in main(), we get this error...

```
NameError: name 'total' is not defined
```

FLYING THROUGH PYTHON

Understanding Global Scope

```
def average(numbers):
    total = 0
    for num in list:
        total = total + num

    avg = total/len(numbers)
    return avg

def main():
    prices = [29, 21, 55, 10]

    result = average(prices)
    print(order_goal)
    print(result)

order_goal = 25
main()
```

Global scope

```
25
28.75
```

If we try to access order_goal in main(), NO errors...

The variables declared outside of a function have **global scope**, so they can be accessed in another function.

FLYING THROUGH PYTHON

Order Matters With Variable Declaration

```
def average(numbers):
    total = 0
    for num in list:
        total = total + num

    avg = total/len(numbers)
    return avg

def main():
    prices = [29, 21, 55, 10]

    result = average(prices)
    print(order_goal)
    print(result)

main()
order_goal = 25
```

At this point, order_goal hasn't been declared yet! So we'll get an error here...

We're calling main() before order_goal is declared...

```
NameError: name 'order_goal' is not defined
```

FLYING THROUGH PYTHON

Level 3

Functions in Flight

Section 3 – Spam Van Functions

Flying through Python

When to Create Functions?

Functions should be used for any chunk of code that has a specific purpose. For instance, here are all of the things we want to implement in our Food Truck Order System:

1. Print the menu
2. Take an order
3. Calculate the total bill

These are all pretty self-contained tasks, so it makes sense that we would create separate functions for them.

We have a plan for the Spam Van — let's get started!

FLYING THROUGH PYTHON

1 Writing a Function to Print Our Menu

```
menu = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50, …}
```

def function(parameters…):

```
def print_menu(menu):
    for name, price in menu.items():
        print(name, ': $', format(price, '.2f'), sep='')
```

parameters: we need the menu to be able to print it

Indented: Contents of the function are indented.

Return value: Functions can return values — this function doesn't need to because it's just printing.

The code to print the menu that we already had

FLYING THROUGH PYTHON

2 Moving the Order Loop to a Function

Function definition

Indented

```
def get_order(menu):
    orders = []
    order = input("What would you like to order? (Q to Quit)")

    while (order.upper() != 'Q'):
        # find the order
        found = menu.get(order)
        if found:
            orders.append(order)
        else:
            print("Menu item doesn't exist")

        order = input("Anything else? (Q to Quit)")

    return orders
```

Return value

Parameters

We need the menu to add the item to the order.

The code to get an order that we already wrote

FLYING THROUGH PYTHON

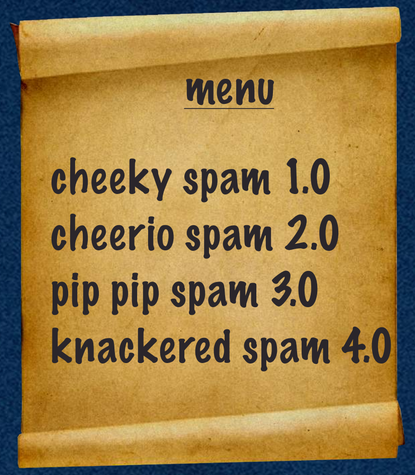
How the main() Function Executes

```
def get_order(menu):
    …

def print_menu(menu):
    …


def main():
    menu = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50, …}
    print_menu(menu)
    order = get_order(menu)
    print("You ordered:",order)

main()
```

FLYING THROUGH PYTHON

3 Calculating the Total Bill

Finally, we want to calculate the customer's total bill.




```
def bill_total(orders, menu):
    total = 0

    for order in orders:
        total += menu[order]

    return total
```

Parameters
We need the list of orders and the menu to look up the price.

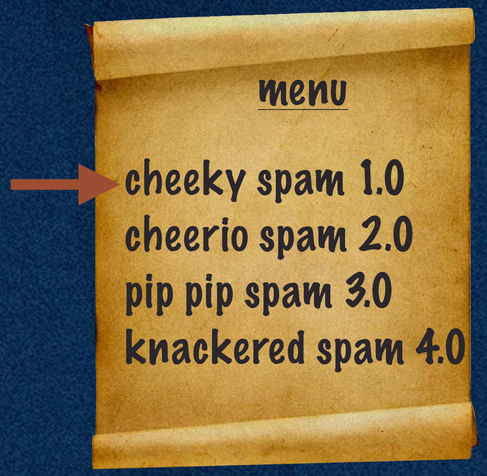
Program logic
For each order in our orders list, we want to add the price to our total.

Return the total

FLYING THROUGH PYTHON

bill_total() After the First Order

Take each order



Look up its price in the menu

First loop:

order: 'cheeky spam'
total = 0 + 1.0 = 1.0

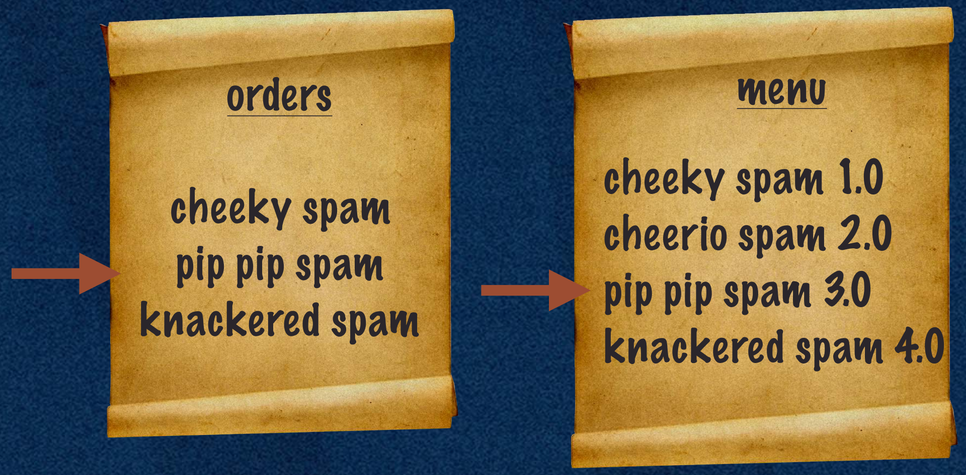
Add the price to the total

```
def bill_total(orders, menu):
    total = 0

    for order in orders:
        total += menu[order]

    return total
```

We look up the price of the item in the menu using the order name as a key.

FLYING THROUGH PYTHON

bill_total() After the Second Order



Take each order

Look up its price in the menu

First loop:

order: 'cheeky spam'
total = 0 + 1.0 = 1.0

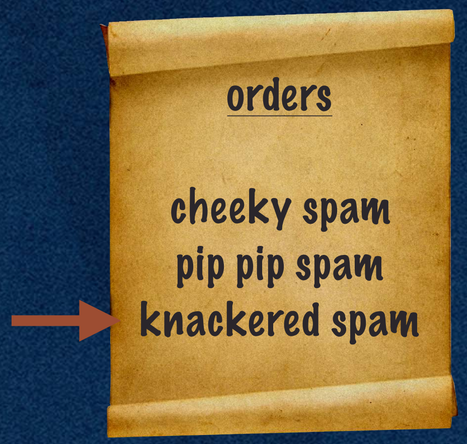
order: 'pip pip spam'
total = 1.0 + 3.0 = 4.0

Add the price to the total

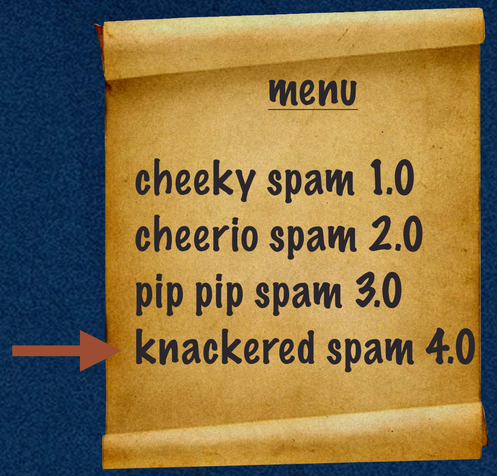
```
def bill_total(orders, menu):
    total = 0

    for order in orders:
        total += menu[order]

    return total
```

FLYING THROUGH PYTHON

bill_total() After the Third Order



Take each order



Look up its price in the menu

First loop:

order: 'cheeky spam'
total = 0 + 1.0 = 1.0

order: 'pip pip spam'
total = 1.0 + 3.0 = 4.0

order: 'knackered spam'
total = 4.0 + 4.0 = 8.0

return total

```
def bill_total(orders, menu):
    total = 0

    for order in orders:
        total += menu[order]

    return total
```

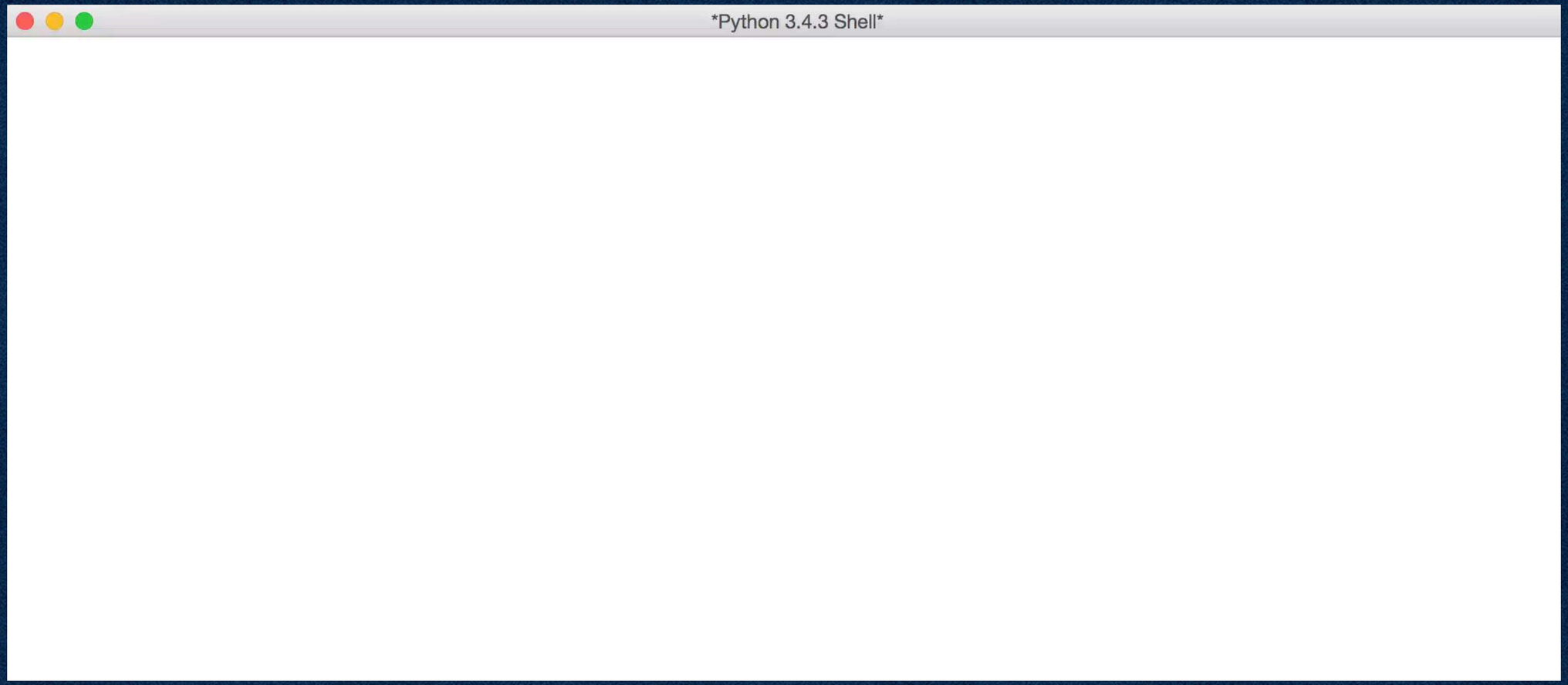
FLYING THROUGH PYTHON

Putting It All Together

spam_van.py

```
def print_menu(menu):
    …

def get_order(menu):
    …

def total_bill(orders, menu):
    …

def main():
    menu = {'Knackered Spam': 0.50, 'Pip pip Spam': 1.50, …}
    print_menu(menu)
    orders = get_order(menu)
    total = bill_total(orders, menu)
    print("You ordered:", order,
          "Your total is: $", format(total,'.2f'), sep='')

main()
```

Demo of the Spam Van



FLYING THROUGH PYTHON

Level 4

Spam Van Data

Section 1 – Writing Files

Flying through Python

Problem: Our Orders Disappear When We Quit

Let's say we have a short program that gets a dictionary of orders from a customer. As soon as the program ends, the dictionary is not in memory anymore...

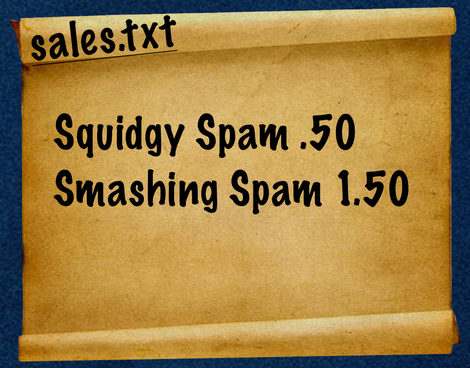
Flying3.py - /Users/sbuchanan/Desktop/Flying3.py (3.4.3)

```
menu = {'Pip pip Spam': 1.50, 'Squidgy Spam': 0.50, 'Smashing Spam': 1.50}

order = {}
item = input("What would you like to order? (Q to Quit)")

while (item.upper() != 'Q'):
    order[item] = menu[item]
    item = input("Anything else? (Q to Quit)")

print('You ordered:', order)
```

Python 3.4.3 Shell

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
===
>>>
What would you like to order? (Q to Quit)
```

Ln: 6 Col: 0



We can save orders by writing our sales to a file.

FLYING THROUGH PYTHON

How to Write to a File

Here are the steps to write to a file.

File name

Mode

1 Open

```
sales_log = open('spam_orders.txt', 'w')
```

2 Write

```
sales_log.write('The Spam Van')
sales_log.write('Sales Log')
```

3 Close

```
sales_log.close()
```

`'w'` for write
`'r'` for read
`'a'` for append

If the file doesn't exist for a write or an append, Python will automatically create the file.

We also need to add our own newlines.

FLYING THROUGH PYTHON

Writing a Newline to a File

Unlike print(), when we use write(), we need to add newlines.

1. Open
2. Write
3. Close

```
sales_log = open('spam_orders.txt', 'w')

sales_log.write('The Spam Van\n')
sales_log.write('Sales Log')

sales_log.close()
```

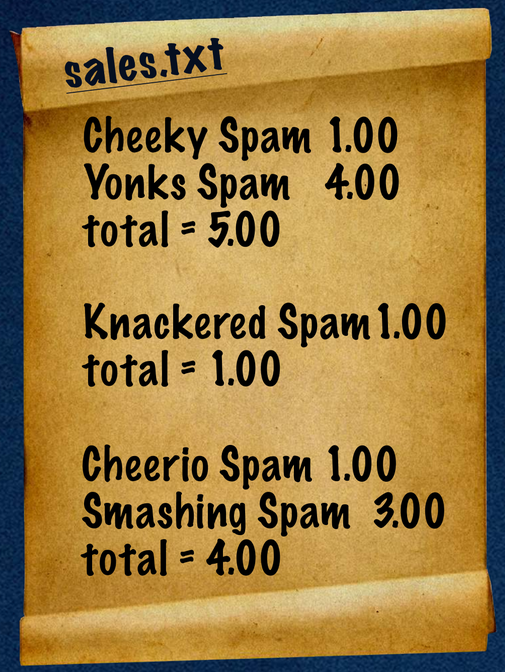
Newline

Newline

FLYING THROUGH PYTHON

We Need to Build a Customer Sales Log

After we get a customer's order, it would be great if we could record each sale to a sales.txt file.



One single order comes in as a dictionary of menu items and their price.

Customer 1

```
order = {'Cheeky Spam': 1.0, 'Chips Spam': 4.0}
```

Customer 2

```
order = {'Knackered Spam': 1.0}
```

Customer 3

```
order = {'Cheerio Spam': 1.0, 'Smashing Spam': 3.0}
```

FLYING THROUGH PYTHON

The Steps to Write to Our Sales Log

One single order

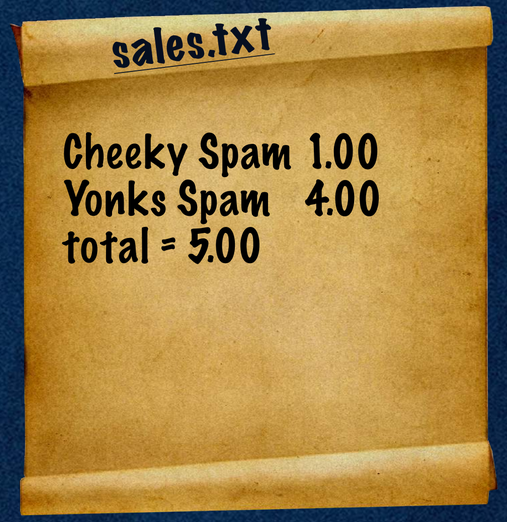
```
order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
```

1 Open

2 Write

3 Close

```
def write_sales_log(order):
    # open the file
    file = open('sales.txt', 'w')


    # write each item to the file
    # write the total to the file


    # close the file
    file.close()
```



FLYING THROUGH PYTHON

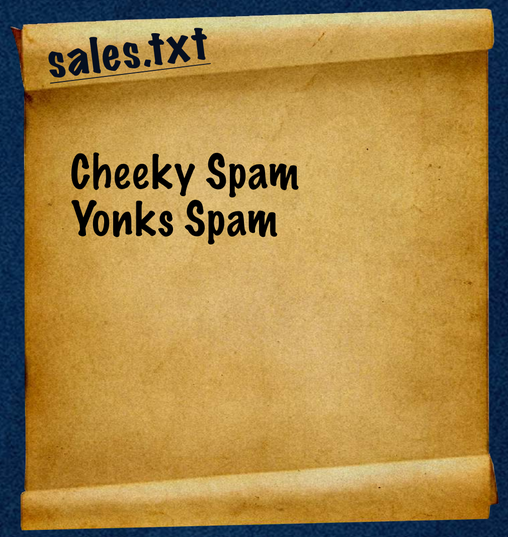
Writing Out Each Order

```
def write_sales_log(order):
    file = open('sales.txt', 'w')

    for order in order:
        file.write(order+'\n')

    file.close()
```

Need order.items() to get the key and value

```
def main():
    order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
    write_sales_log(order)
```

We'll call our write_sales_log() function from our main() function.



FLYING THROUGH PYTHON

Writing Out the Price

```
def write_sales_log(order):
    file = open('sales.txt', 'w')

    for item, price in order.items():
        file.write(item + ' ' + format(price, '.2f') + '\n')

    file.close()
```

```
def main():
    order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
    write_sales_log(order)
```

We also want the order total

FLYING THROUGH PYTHON

Writing Out the Order Total

Create the total

Add to it

Write it

```
def write_sales_log(order):
    file = open('sales.txt', 'w')

    total = 0
    for item, price in order.items():
        file.write(item + ' ' + format(price, '.2f') + '\n')
        total += price

    file.write('total = ' + format(total, '.2f') + '\n')
    file.close()
```

```
def main():
    order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
    write_sales_log(order)
```

Now our sales log has the total of all orders

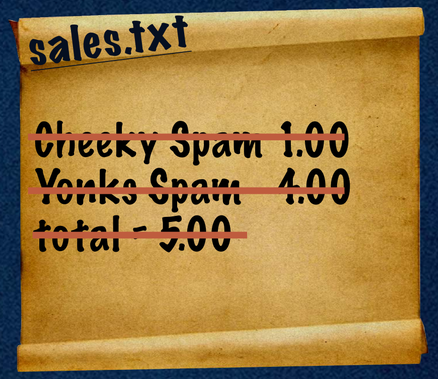
FLYING THROUGH PYTHON

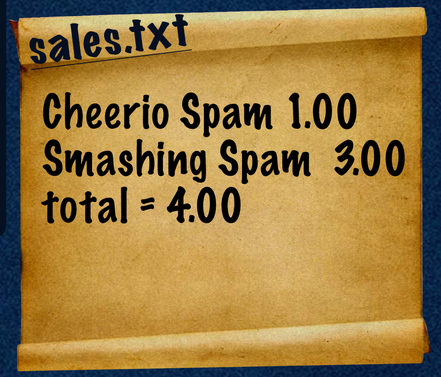
We Don't Want to Overwrite Our File

```
def write_sales_log(order):
    file = open('sales.txt', 'w')

    total = 0
    for item, price in order.items():
        file.write(item + ' ' + format(price, '.2f') + '\n')
        total += price

    file.write('total = ' + format(total, '.2f') + '\n')
    file.close()
```

Change this to 'a' for append

```
def main():
    order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
    write_sales_log(order)
    order = {'Cheerio Spam': 1.0, 'Smashing Spam': 3.0}
    write_sales_log(order)
```





We overwrite the file...
Instead, we want to append.

FLYING THROUGH PYTHON

Appending Orders to Our File

```
def write_sales_log(order):
    file = open('sales.txt', 'a' )

    total = 0
    for item, price in order.items():
        file.write(item + ' ' + format(price, '.2f') + '\n')
        total += price

    file.write('total = ' + format(total, '.2f') + '\n')
    file.close()
```

'a' for append

```
def main():
    order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
    write_sales_log(order)
    order = {'Cheerio Spam': 1.0, 'Smashing Spam': 3.0}
    write_sales_log(order)
```

sales.txt

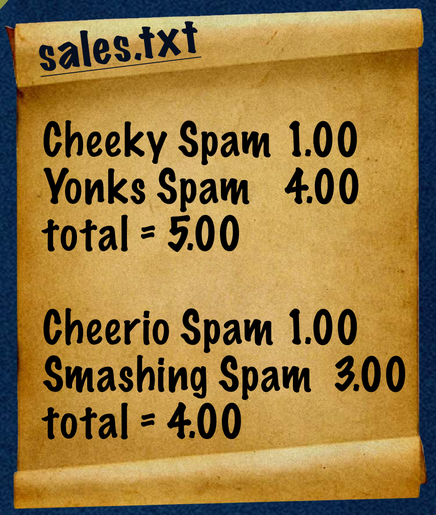
```
Cheeky Spam 1.00
Yonks Spam  4.00
total = 5.00
Cheerio Spam 1.00
Smashing Spam 3.00
total = 4.00
```

FLYING THROUGH PYTHON

Adding a Newline Between Orders

```
def write_sales_log(order):
    file = open('sales.txt','a' )

    total = 0
    for item, price in order.items():
        file.write(item + ' ' + format(price, '.2f') + '\n')
        total += price

    file.write('total = ' + format(total, '.2f') + '\n\n')
    file.close()
```

```
def main():
    order = {'Cheeky Spam': 1.0, 'Yonks Spam': 4.0}
    write_sales_log(order)
    order = {'Cheerio Spam': 1.0, 'Smashing Spam': 3.0}
    write_sales_log(order)
```



FLYING THROUGH PYTHON

Level 4

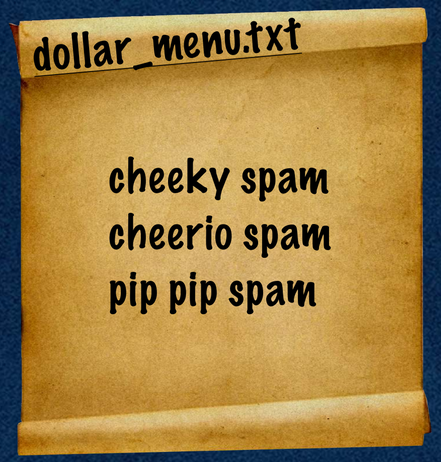
Spam Van Data

Section 2 – Reading Files

Flying through Python

The Spam Van Has a Value Menu

Every day, the boss sends us a file of dollar menu items. We want to read this file into a list so our program can use it.

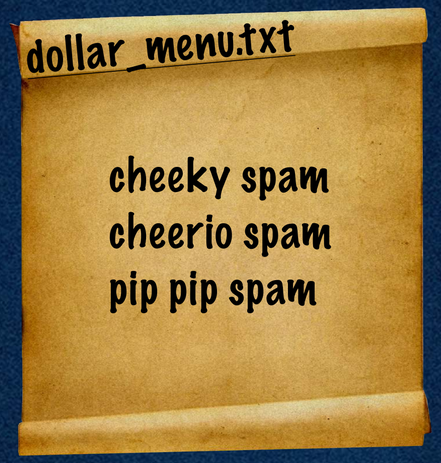



```
['Cheeky Spam', 'Cheerio Spam', 'Pip Pip Spam']
```

FLYING THROUGH PYTHON

Reading the Entire Contents of a File

`file_name.read()` will return a string containing the entire contents of the file.



file name

mode

1 open

2 read

3 close

```
dollar_spam = open('dollar_menu.txt', 'r')
print(dollar_spam.read())
dollar_spam.close()
```

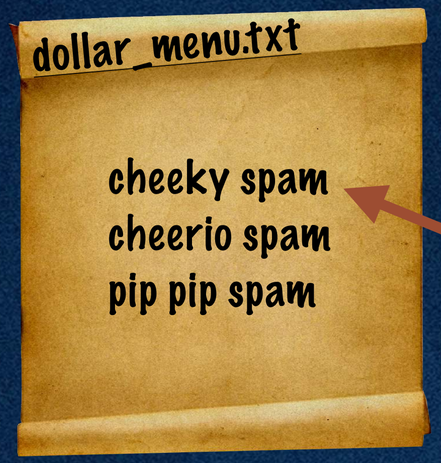
'r' for read

```
cheeky spam
cheerio spam
pip pip spam
```

FLYING THROUGH PYTHON

Reading an Individual Line From a File

`file_name.readline()` will return a string for the next single line in the file.



readline() will read until a newline '\n'

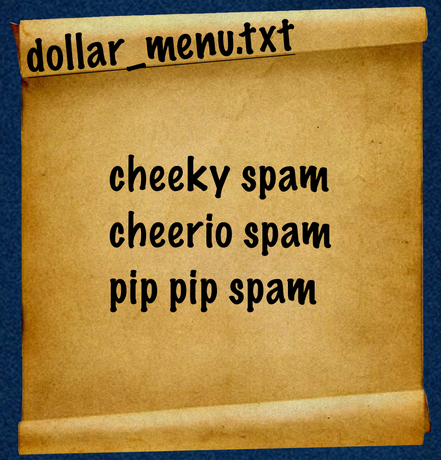
```
def read_dollar_menu():
    dollar_spam = open('dollar_menu.txt', 'r')
    print('1st line:', dollar_spam.readline())
    print('2nd line:', dollar_spam.readline())
    dollar_spam.close()
```

```
1st line: cheeky spam
2nd line: cheerio spam
```

Flying through Python

Reading the Dollar Menu as a List of Lines

We can read all of the lines individually in a **loop**, which will be convenient to add each item to a list.




```
def read_dollar_menu():
    dollar_spam = open('dollar_menu.txt', 'r')

    for line in dollar_spam:
        print(line)

    dollar_spam.close()
```

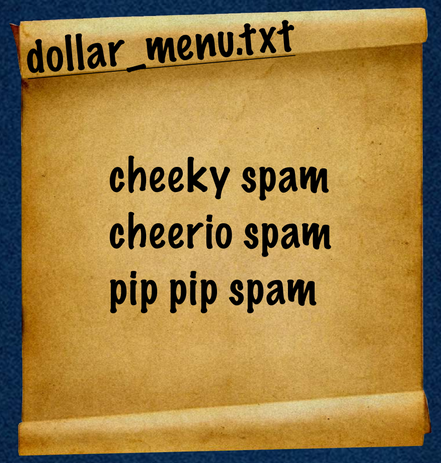
Now, instead of printing each line, let's add them to a list.

```
Cheerio Spam

Pip Pip Spam

Knackered Spam

Cheeky Spam
```

FLYING THROUGH PYTHON

Reading the Dollar Menu Into a List

We can add each line item to a list inside of our for loop.




```
def read_dollar_menu():
    dollar_spam = open('dollar_menu.txt', 'r')

    dollar_menu = []
    for line in dollar_spam:
        dollar_menu.append(line)

    print(dollar_menu)
    dollar_spam.close()
```

This will combine all items into a list.

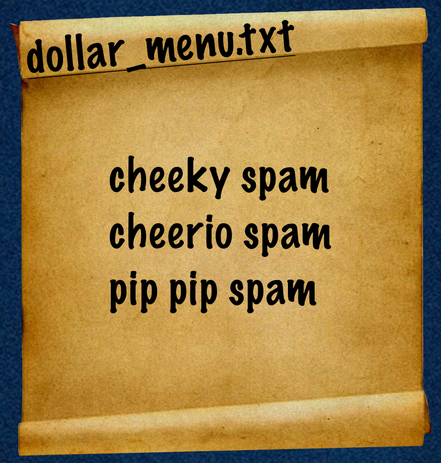
```
['Cheerio Spam\n', 'Pip Pip Spam\n', 'Knackered Spam\n', 'Cheeky Spam']
```

We want to get rid of the newlines \n.

FLYING THROUGH PYTHON

Stripping Leading or Trailing Whitespace

Since each line has the newline character at the end, we want to strip that off.




```
def read_dollar_menu():
    dollar_spam = open('dollar_menu.txt', 'r')

    dollar_menu = []
    for line in dollar_spam:
        line = line.strip()
        dollar_menu.append(line)

    print(dollar_menu)
    dollar_spam.close()
```

str.strip() removes any leading or trailing whitespace.

```
['Cheerio Spam', 'Pip Pip Spam', 'Knackered Spam', 'Cheeky Spam']
```

Newlines are gone!

FLYING THROUGH PYTHON

Level 4

Spam Van Data

Section 3 – Try, Except

Flying through Python

Problem: The Program Crashes When an Error Happens

What happens if we try to open a file that doesn't exist?

```
my_file = open('name.txt', 'r')
```

```
FileNotFoundError: No such file or directory: 'name.txt'
```

This error will cause our program to crash...

It would be great if we could recover and continue our program even if we get an error!

Flying through Python

Using Try, Except to Recover From Errors

Try, except blocks catch errors so that we can avoid a program crash.

```
try:
    file = open('sales.txt', 'r')
    print(file.read())
except:
    print('File doesn't exist')
```

Like saying "try this"
and if you get an error...

Go here and print this error message

And then continue with the program

```
File doesn't exist
```

Program prints your error message,
telling the user what the problem was.

FLYING THROUGH PYTHON

Types of Exceptions

Python has 60-plus types of exceptions. Here are some of the ones we've seen before:

`FileNotFoundError`	File doesn't exist
`IndexError`	Index out of bounds
`KeyError`	Key doesn't exist
`NameError`	Variable name doesn't exist in local or global scope
`ValueError`	Value is the wrong type

Let's look at specifically handling this value error next...

You can find all of the types of
Python 3 exceptions here: http://go.codeschool.com/python-exceptions

FLYING THROUGH PYTHON

Try, Except in Other Situations

We can also use try, except in other situations that might have unexpected problems...
like casting user input to a number.

```
price = input("Enter the price: ")

try:
    price = float(price)
    print('Price =', price)
except ValueError:
    print('Not a number!')
```

We can also look for a specific type of error.

```
Enter the price: 25
Price = 25.0
```

The user enters a number so there's no error.

```
Enter the price: oops
Not a number!
```

Program prints the exception, telling the user what the problem was, then continues.

FLYING THROUGH PYTHON

Storing the Exception's Error Message

We can also save the exception's error message in a variable for printing.

```
price = input("Enter the price: ")

try:
    price = float(price)
    print('Price =', price)
except ValueError as err:
    print(err)
```

Create a variable 'err' to hold the error message.

```
Enter the price: yo
could not convert string to float: 'yo'
```

We can just print out the exact same error the Python interpreter gives us.

FLYING THROUGH PYTHON

Level 5

Help Me Help You

Section 1 – Using Modules

Flying Through Python

The Spam Van Franchise

The Spam Van is now a franchise, and each of the vans download the daily menu from a website.

To do this, we need
the Python requests module.

FLYING THROUGH PYTHON

Introducing Modules

Modules are code libraries that contain functions we can call from our code. They make our lives easier by implementing the hard stuff for us.

Instead of rewriting complicated code, it's already been done for us!

```
import random


ticket = random.randint(1, 1000)
print(ticket)
```

Reuse code instead of reinventing the wheel!

```
import math


answer = math.sqrt(3)
print(answer)
```

Modules — another tool for the toolbox!

FLYING THROUGH PYTHON

Installing Modules With Pip

The requests module does not come pre-installed with Python, so we'll have to install it. Pip is a package manager that will install modules for us.

pip install
installs python modules

Name of the module

```
>>> pip install requests
```

```
Collecting requests
downloading …100%
Installing collected packages: requests
Successfully installed requests-2.8.1
```

pip is a package manager that comes pre-installed with Python 3.4.
For earlier versions, you can install pip with: `python get-pip.py`

Check out all available Python packages: http://go.codeschool.com/pypi

FLYING THROUGH PYTHON

The HTTP Request Response Cycle

spam_van.py

Our request for data from the server is called an HTTP request.

The server responds with data in a format called JSON.

Web Server

FLYING THROUGH PYTHON

Using JSON to Format and Share Data

JSON is a standard way to format and share data.

JSON uses a collection of keys and values, which look just like Python dictionaries ...

Dictionary
{'key': 'value', 'key': 'value'}

```
[
  {
    "price": "3.00",
    "name": "Omelet",
    "desc": "Yummy"
  },
  {
    "price": "5.75",
    "name": "Burrito",
    "desc": "Breakfast Burrito"
  },
  {
    "price": "4.50",
    "name": "Waffles",
    "desc": "Belgian waffles with syrup"
  }
]
```

FLYING THROUGH PYTHON

JSON Looks Like Dictionaries and Lists

JSON uses a collection of keys and values, which look just like Python dictionaries and lists

List
[...]

Dictionary
{'key': 'value', 'key': 'value'}

```
[
    {
      "price": "3.00",
      "name": "Omelet",
      "desc": "Yummy"
    },
    {
      "price": "5.75",
      "name": "Burrito",
      "desc": "Breakfast Burrito"
    },
    {
      "price": "4.50",
      "name": "Waffles",
      "desc": "Belgian waffles with syrup"
    }
]
```

Flying through Python

How Would JSON Be Used to Store Our Menu?

What if we wanted to store more information about a menu item — its name, description, and price? We could do this with a dictionary that has keys for name, description, and price.

```
first_item  = {'name': 'Spam n Eggs',
               'description': 'Two eggs with Spam',
               'price':2.50}

second_item = {'name': 'Spam n Jam',
               'description': 'Biscuit with Jam with Spam',
               'price':3.50}

print(first_item['name'],  first_item['price'])
print(second_item['name'], second_item['price'])
```

```
Spam n Eggs 2.5
Spam n Jam 3.5
```

FLYING THROUGH PYTHON

A List of Dictionaries

```
first_item  = {'name': 'Spam n Eggs',
               'description': 'Two eggs with Spam',
               'price':2.50}

second_item = {'name': 'Spam n Jam',
               'description': 'Biscuit with Jam with Spam',
               'price':3.50}

menu_items = [first_item, second_item]

print(menu_items[0]['name'], menu_items[0]['price'], menu_items[0]['desc'])
print(menu_items[1]['name'], menu_items[1]['price'], menu_items[1]['desc'])
```

Remember: first access the outer list, then access the dictionary key.

```
Spam n Eggs 2.5 Two eggs & Spam
Spam n Jam 3.5 Biscuit, Jam & Spam
Spam n Ham 4.5 Ham & Spam
```

FLYING THROUGH PYTHON

A List of Dictionaries That Looks Like JSON

```
menu_items = [{'name': 'Spam n Eggs',
               'desc': 'Two eggs & Spam',
               'price':2.50},
              {'name': 'Spam n Jam',
               'desc': 'Biscuit, Jam & Spam',
               'price':3.50},
              {'name': 'Spam n Ham',
               'desc': 'Ham & Spam',
               'price':4.50}]
```

menu_items[0]

menu_items[1]

menu_items[2]

Now our format is the same as the JSON data we'll get from our request.

Instead of hard-coding this menu, let's get it from our HTTP request.

FLYING THROUGH PYTHON

Getting Today's Menu

Requesting and printing today's menu.

```
import requests

my_request = requests.get('http://go.codeschool.com/spamvanmenu')

menu_list = my_request.json()

print(menu_list)
```

Import the module so we can use it for requests

Call requests.get() with our URL

Get the response in JSON format

```
[{"price": "3.00", "name": "Omelet",  "desc": "Yummy"},
 {"price": "5.75", "name": "Burrito", "desc": "Breakfast Burrito"},
 {"price": "4.50", "name": "Waffles", "desc": "Belgian waffles with syrup"}]
```

FLYING THROUGH PYTHON

Printing Out the Menu

Requesting and printing today's menu.

```
import requests

my_request = requests.get('http://go.codeschool.com/spamvanmenu')

menu_list = my_request.json()

print("Today's Menu:")
for item in menu_list:
    print(item['name'], item['desc'], item['price'])
```

```
Today's Menu:
Omelet Yummy 3.00
Burrito Breakfast Burrito 5.75
Waffles Belgian waffles with syrup 4.50
```

FLYING THROUGH PYTHON

Formatting the Menu

Requesting and printing today's menu.

```
import requests

my_request = requests.get('http://go.codeschool.com/spamvanmenu')

menu_list = my_request.json()

print("Today's Menu:")
for item in menu_list:
    print(item['name'], ': ', item['desc'].title(), ', $',
          item['price'],  sep='')
```

```
Today's Menu:
Omelet: Yummy, $3.00
Burrito: Breakfast Burrito, $5.75
Waffles: Belgian waffles with syrup, $4.50
```

FLYING THROUGH PYTHON

Demo

spam_van.py - /Users/sbuchanan/Desktop/spam_van.py (3.4.3)

```
import requests

my_request = requests.get('http://go.codeschool.com/spamvanmenu')
menu_list = my_request.json()

print("Today's Menu:")
for item in menu_list:
    print(item['name'], ':  ', item['desc'].title(), ', $',
          item['price'],  sep='')
```

Python 3.4.3 Shell

Level 5

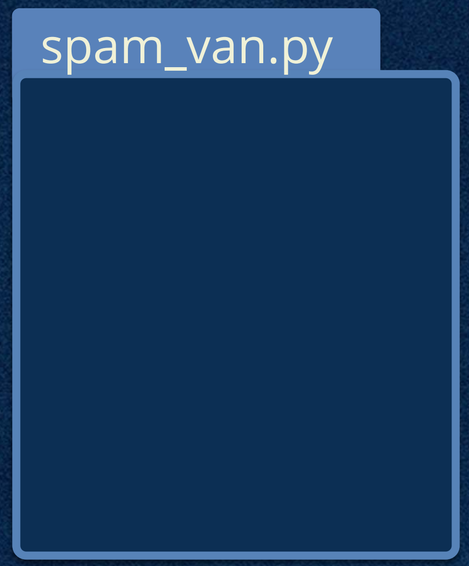
Help Me Help You

Section 2 – Creating Modules

Flying through Python

Lots of Code in One File Is a Problem

If we put all of our code in one file, it's going to get really hard to read and maintain.



Right now, this one script is responsible for:

- printing the menu
- taking orders
- sales reports
- accessing the menu from the network

Flying through Python

Creating Modules for Organization

A script containing definitions is a ***module*** and can be imported by a script or another module.

spam_van.py

```
import orders
import menu
import sales
```

Script

orders.py

Module

menu.py

Module

sales.py

Module

same directory

As long as all of these files are in the same directory, we can import them into spam_van.py by writing `import module_name`.

FLYING THROUGH PYTHON

Figuring Out What Goes in the Orders Module

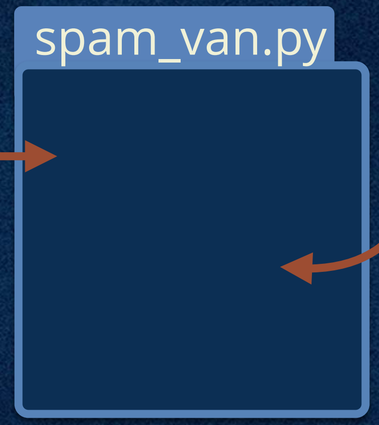
We want to move these definitions into a separate orders.py file.

Our main functionality stays in our spam_van.py script.

spam_van.py

```
def print_menu(menu):
    …
def get_order(menu):
    …
def total_bill(orders, menu):
    …

def main():
    menu = {'Cheerio Spam': 0.50,…}
    print_menu(menu)
    orders = get_order(menu)
    total = bill_total(orders, menu)
    print('You ordered:', order, 'Total:', total)

main()
```

Need to figure out how to use orders functionality in here



The Result of Splitting Orders Into a Module

orders.py

```
def print_menu(menu):
    …
def get_order(menu):
    …
def total_bill(orders, menu):
    …
```

spam_van.py

```
def main():
    menu = {'Cheerio Spam': 0.50,…}
    print_menu(menu)
    orders = get_order(menu)
    total = bill_total(orders, menu)
    print('You ordered:', order, 'Total:', total)

main()
```

When we run spam_van.py, we get this error.

```
NameError: name 'print_menu' is not defined
```

Importing the Orders Module

orders.py

```
def print_menu(menu):
    …
def get_order(menu):
    …
def total_bill(orders, menu):
    …
```

spam_van.py

```
import orders

def main():
    menu = {'Cheerio Spam': 0.50,…}
    print_menu(menu)
    orders = get_order(menu)
    total = bill_total(orders, menu)
    print('You ordered:', order, 'Total:', total)

main()
```

This import will find our orders.py file if it's in the same directory.

Wait — this is still generating an error.

```
NameError: name 'print_menu' is not defined
```

Using the Module Name to Access the Functions

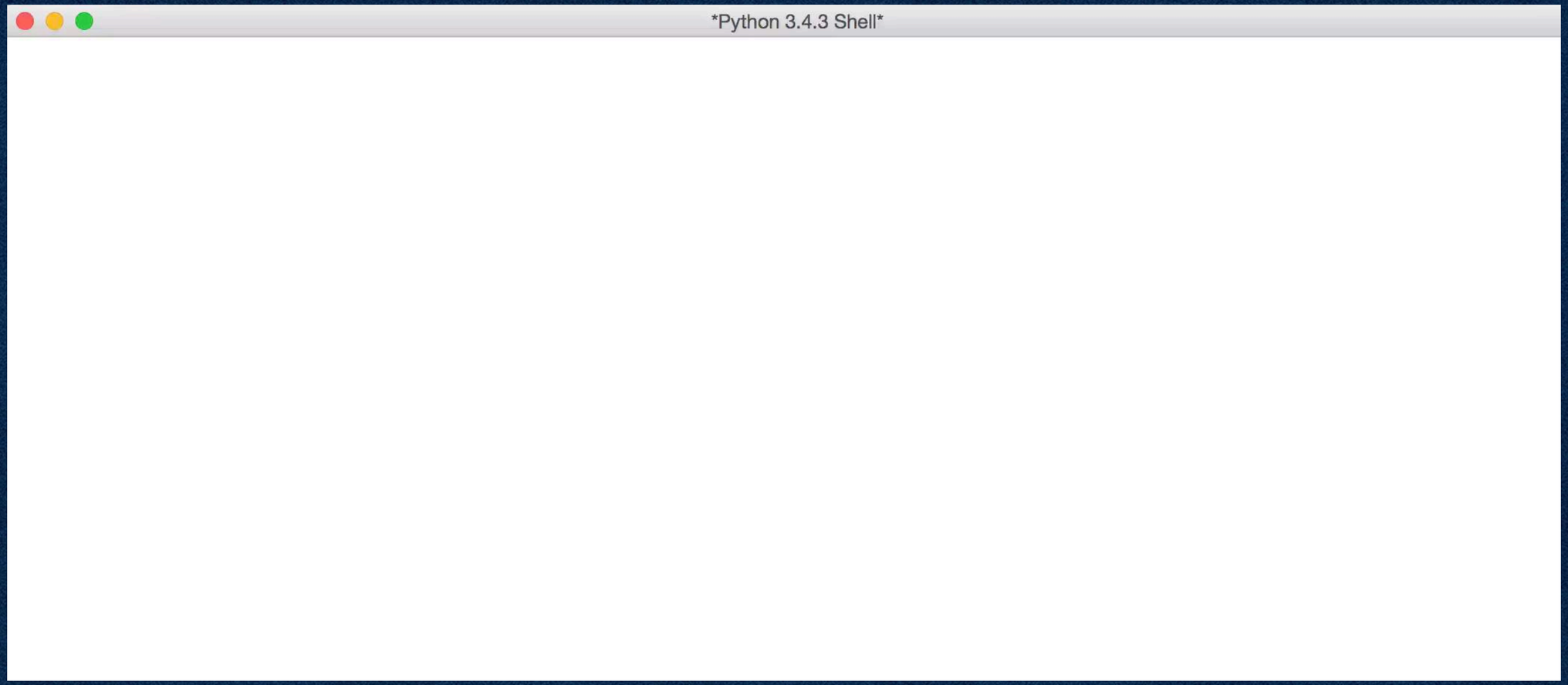
orders.py

```
def print_menu(menu):
    …
def get_order(menu):
    …
def total_bill(orders, menu):
    …
```

spam_van.py

```
import orders

def main():
    menu = {'Cheerio Spam': 0.50,…}
    orders.print_menu(menu)
    orders = orders.get_order(menu)
    total = orders.bill_total(orders, menu)
    print('You ordered:', order, 'Total:', total)

main()
```

We need to add the module name orders. before our function calls

Now our program will run!

Demo: The App Running