# Appendix D. Answers to Exercises

## 1. A Taste of Py

1.1 If you don't already have Python 3 installed on your computer, do it now. Read Appendix B for the details for your computer system.

1.2 Start the Python 3 interactive interpreter. Again, details are in Appendix B. It should print a few lines about itself and then a single line starting with `>>>`. That's your prompt to type Python commands.

---

Here's what it looks like on my Mac:

```
$ python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:39:00)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

---

1.3 Play with the interpreter a little. Use it like a calculator and type this: `8 * 9`. Press the Enter key to see the result. Python should print `72`.

---

```
>>> 8 * 9
72
```

---

1.4 Type the number `47` and press the Enter key. Did it print 47 for you on the next line?

```
>>> 47
47
```

---

1.5 Now type `print(47)` and press Enter. Did that also print 47 for you on the next line?

---

```
>>> print(47)
47
```

---

# 2. Data: Types, Values, Variables, and Names

2.1 Assign the integer value `99` to the variable `prince`, and print it.

---

```
>>> prince = 99
>>> print(prince)
99
>>>
```

---

2.2 What type is the value `5`?

---

```
>>> type(5)
<class 'int'>
```

---

2.3 What type is the value `2.0`?

---

```
>>> type(2.0)
<class 'float'>
```

---

2.4 What type is the expression `5 + 2.0`?

---

```
>>> type(5 + 2.0)
<class 'float'>
```

---

# 3. Numbers

3.1 How many seconds are in an hour? Use the interactive interpreter as a calculator and multiply the number of seconds in a minute ( `60` ) by the number of minutes in an hour (also `60`).

---

```
>>> 60 * 60
3600
```

---

3.2 Assign the result from the previous task (seconds in an hour) to a variable called `seconds_per_hour`.

```
>>> seconds_per_hour = 60 * 60
>>> seconds_per_hour
3600
```

---

3.3 How many seconds are in a day? Use your `seconds_per_hour` variable.

---

```
>>> seconds_per_hour * 24
86400
```

---

3.4 Calculate seconds per day again, but this time save the result in a variable called `seconds_per_day`.

---

```
>>> seconds_per_day = seconds_per_hour * 24
>>> seconds_per_day
86400
```

---

3.5 Divide `seconds_per_day` by `seconds_per_hour`. Use floating-point (`/`) division.

---

```
>>> seconds_per_day / seconds_per_hour
24.0
```

---

3.6 Divide `seconds_per_day` by `seconds_per_hour`, using integer (`//`) division. Did this number agree with the floating-point value from the

previous question, aside from the final `.0`?

---

```
>>> seconds_per_day // seconds_per_hour
24
```

---

# 4. Choose with if

4.1 Choose a number between 1 and 10 and assign it to the variable `secret`. Then, select another number between 1 and 10 and assign it to the variable `guess`. Next, write the conditional tests (`if`, `else`, and `elif`) to print the string `'too low'` if `guess` is less than `secret`, `'too high'` if greater than `secret`, and `'just right'` if equal to `secret`.

---

Did you choose `7` for `secret`? I bet a lot of people do, because there's something about `7`.

```
secret = 7
guess = 5
if guess < secret:
    print('too low')
elif guess > secret:
    print('too high')
else:
    print('just right')
```

Run this program and you should see the following:

```
too low
```

---

4.2 Assign `True` or `False` to the variables `small` and `green`. Write some `if`/`else` statements to print which of these matches those choices:

cherry, pea, watermelon, pumpkin.

---

```
>>> small = False
>>> green = True
>>> if small:
...     if green:
...         print("pea")
...     else:
...         print("cherry")
... else:
...     if green:
...         print("watermelon")
...     else:
...         print("pumpkin")
...
watermelon
```

---

# 5. Text Strings

5.1 Capitalize the word starting with `m`:

```
>>> song = """When an eel grabs your arm,
... And it causes great harm,
... That's - a moray!"""
```

Don't forget the space before the `m`:

```
>>> song = """When an eel grabs your arm,
... And it causes great harm,
... That's - a moray!"""
>>> song = song.replace(" m", " M")
>>> print(song)
When an eel grabs your arm,
And it causes great harm,
That's - a Moray!
```

---

5.2 Print each list question with its correctly matching answer, in the form:

Q: *question*
A: *answer*

```
>>> questions = [
...     "We don't serve strings around here. Are you a string?",
...     "What is said on Father's Day in the forest?",
...     "What makes the sound 'Sis! Boom! Bah!'?"
...     ]
>>> answers = [
...     "An exploding sheep.",
...     "No, I'm a frayed knot.",
...     "'Pop!' goes the weasel."
...     ]
```

You could print each item in `questions` with its mate from `answers` in many ways. Let's try a tuple sandwich (tuples in a tuple) to pair them, and tuple unpacking to retrieve them for printing:

```
questions = [
    "We don't serve strings around here. Are you a string?",
    "What is said on Father's Day in the forest?",
    "What makes the sound 'Sis! Boom! Bah!'?"
    ]
answers = [
    "An exploding sheep.",
    "No, I'm a frayed knot.",
    "'Pop!' goes the weasel."
    ]

q_a = ( (0, 1), (1,2), (2, 0) )
for q, a in q_a:
    print("Q:", questions[q])
    print("A:", answers[a])
    print()
```

Output:

```
$ python qanda.py
Q: We don't serve strings around here. Are you a string?
A: No, I'm a frayed knot.

Q: What is said on Father's Day in the forest?
A: 'Pop!' goes the weasel.

Q: What makes the sound 'Sis! Boom! Bah!'?
A: An exploding sheep.
```

---

5.3 Write the following poem by using old-style formatting. Substitute the strings `'roast beef'`, `'ham'`, `'head'`, and `'clam'` into this string:

```
My kitty cat likes %s,
```

```
My kitty cat likes %s,
My kitty cat fell on his %s
And now thinks he's a %s.
```

---

```
>>> poem = '''
... My kitty cat likes %s,
... My kitty cat likes %s,
... My kitty cat fell on his %s
... And now thinks he's a %s.
... '''
>>> args = ('roast beef', 'ham', 'head', 'clam')
>>> print(poem % args)

My kitty cat likes roast beef,
My kitty cat likes ham,
My kitty cat fell on his head
And now thinks he's a clam.
```

---

5.4 Write a form letter by using new-style formatting. Save the following string as `letter` (you'll use it in the next exercise):

```
Dear {salutation} {name},

Thank you for your letter. We are sorry that our {product}
{verbed} in your {room}. Please note that it should never
be used in a {room}, especially near any {animals}.

Send us your receipt and {amount} for shipping and handling.
We will send you another {product} that, in our tests,
is {percent}% less likely to have {verbed}.

Thank you for your support.

Sincerely,
{spokesman}
{job_title}
```

```
>>> letter = '''
... Dear {salutation} {name},
...
... Thank you for your letter. We are sorry that our {product}
... {verbed} in your {room}. Please note that it should never
... be used in a {room}, especially near any {animals}.
...
... Send us your receipt and {amount} for shipping and handling.
... We will send you another {product} that, in our tests,
... is {percent}% less likely to have {verbed}.
...
... Thank you for your support.
...
... Sincerely,
... {spokesman}
... {job_title}
... '''
```

---

5.5 Assign values to variable strings named `'salutation'`, `'name'`, `'product'`, `'verbed'` (past tense verb), `'room'`, `'animals'`, `'percent'`, `'spokesman'`, and `'job_title'`. Print `letter` with these values, using `letter.format()`.

```
>>> print (
...     letter.format(salutation='Ambassador',
...                   name='Nibbler',
...                   product='pudding',
...                   verbed='evaporated',
...                   room='gazebo',
...                   animals='octothorpes',
...                   amount='$1.99',
...                   percent=88,
...                   spokesman='Shirley Iugeste',
...                   job_title='I Hate This Job')
...     )

Dear Ambassador Nibbler,

Thank you for your letter. We are sorry that our pudding
evaporated in your gazebo. Please note that it should never
be used in a gazebo, especially near any octothorpes.

Send us your receipt and $1.99 for shipping and handling.
We will send you another pudding that, in our tests,
is 88% less likely to have evaporated.

Thank you for your support.

Sincerely,
Shirley Iugeste
I Hate This Job
```

---

5.6 After public polls to name things, a pattern emerged: an English submarine (Boaty McBoatface), an Australian racehorse (Horsey McHorseface), and a Swedish train (Trainy McTrainface). Use `%` formatting to print the winning name at the state fair for a prize duck, gourd, and spitz.

**Example D-1. mcnames1.py**

```
names = ["duck", "gourd", "spitz"]
for name in names:
    cap_name = name.capitalize()
    print("%sy Mc%sface" % (cap_name, cap_name))
```

Output:

```
Ducky McDuckface
Gourdy McGourdface
Spitzy McSpitzface
```

---

5.7 Do the same, with `format()` formatting.

---

**Example D-2. mcnames2.py**

```
names = ["duck", "gourd", "spitz"]
for name in names:
    cap_name = name.capitalize()
    print("{}y Mc{}face".format(cap_name, cap_name))
```

---

5.8 Once more, with feeling, and *f strings*.

---

**Example D-3. mcnames3.py**

```
names = ["duck", "gourd", "spitz"]
for name in names:
    cap_name = name.capitalize()
    print(f"{cap_name}y Mc{cap_name}face")
```

---

# 6. Loop with while and for

6.1 Use a `for` loop to print the values of the list `[3, 2, 1, 0]`.

---

```
>>> for value in [3, 2, 1, 0]:
...     print(value)
...
3
2
1
0
```

---

6.2 Assign the value `7` to the variable `guess_me`, and the value `1` to the variable `number`. Write a `while` loop that compares `number` with `guess_me`. Print `'too low'` if `number` is less than `guess me`. If `number` equals `guess_me`, print `'found it!'` and then exit the loop. If `number` is greater than `guess_me`, print `'oops'` and then exit the loop. Increment `number` at the end of the loop.

```
guess_me = 7
number = 1
while True:
    if number < guess_me:
        print('too low')
    elif number == guess_me:
        print('found it!')
        break
    elif number > guess_me:
        print('oops')
        break
    number += 1
```

If you did this right, you should see this:

```
too low
too low
too low
too low
too low
too low
found it!
```

Notice that the `elif start > guess_me:` line could have been a simple `else:`, because if `start` is not less than or equal to `guess_me`, it must be greater—at least in this universe.

---

6.3 Assign the value `5` to the variable `guess_me`. Use a `for` loop to iterate a variable called `number` over `range(10)`. If `number` is less than `guess_me`, print `'too low'`. If it equals `guess_me`, print `found it!` and then break out of the for loop. If `number` is greater than `guess_me`, print `'oops'` and then exit the loop.

```
>>> guess_me = 5
>>> for number in range(10):
...     if number < guess_me:
...         print("too low")
...     elif number == guess_me:
...         print("found it!")
...         break
...     else:
...         print("oops")
...         break
...
too low
too low
too low
too low
too low
found it!
```

---

# 7. Tuples and Lists

7.1 Create a list called `years_list`, starting with the year of your birth, and each year thereafter until the year of your fifth birthday. For example, if you were born in 1980, the list would be `years_list = [1980, 1981, 1982, 1983, 1984, 1985]`.

---

If you were born in 1980, you would type:

```
>>> years_list = [1980, 1981, 1982, 1983, 1984, 1985]
```

---

7.2 In which of these years was your third birthday? Remember, you were 0 years of age for your first year.

You want offset `3`. Thus, if you were born in 1980:

```
>>> years_list[3]
1983
```

---

7.3 In which year in `years_list` were you the oldest?

---

You want the last year, so use offset `-1`. You could also say `5` because you know this list has six items, but `-1` gets the last item from a list of any size. For a 1980-vintage person:

```
>>> years_list[-1]
1985
```

---

7.4 Make and print a list called `things` with these three strings as elements: `"mozzarella"`, `"cinderella"`, `"salmonella"`.

---

```
>>> things = ["mozzarella", "cinderella", "salmonella"]
>>> things
['mozzarella', 'cinderella', 'salmonella']
```

---

7.5 Capitalize the element in `things` that refers to a person and then print the list. Did it change the element in the list?

This capitalizes the word but doesn't change it in the list:

```
>>> things[1].capitalize()
'Cinderella'
>>> things
['mozzarella', 'cinderella', 'salmonella']
```

If you want to change it in the list, you need to assign it back:

```
>>> things[1] = things[1].capitalize()
>>> things
['mozzarella', 'Cinderella', 'salmonella']
```

---

7.6 Make the cheesy element of `things` all uppercase and then print the list.

---

```
>>> things[0] = things[0].upper()
>>> things
['MOZZARELLA', 'Cinderella', 'salmonella']
```

---

7.7 Delete the disease element, collect your Nobel Prize, and then print the list.

This would remove it by value:

```
>>> things.remove("salmonella")
>>> things
['MOZZARELLA', 'Cinderella']
```

Because it was last in the list, the following would have worked also:

```
>>> del things[-1]
```

And you could have deleted by offset from the beginning:

```
>>> del things[2]
```

---

7.8 Create a list called `surprise` with the elements `"Groucho"`, `"Chico"`, and `"Harpo"`.

---

```
>>> surprise = ['Groucho', 'Chico', 'Harpo']
>>> surprise
['Groucho', 'Chico', 'Harpo']
```

---

7.9 Lowercase the last element of the `surprise` list, reverse it, and then capitalize it.

---

```
>>> surprise[-1] = surprise[-1].lower()
>>> surprise[-1] = surprise[-1][::-1]
>>> surprise[-1].capitalize()
'Oprah'
```

---

7.10 Use a list comprehension to make a list called `even` of the even numbers in `range(10)`.

---

```
>>> even = [number for number in range(10) if number % 2 == 0]
>>> even
[0, 2, 4, 6, 8]
```

---

7.11 Let's create a jumprope rhyme maker. You'll print a series of two-line rhymes. Start with this program fragment:

```
start1 = ["fee", "fie", "foe"]
rhymes = [
    ("flop", "get a mop"),
    ("fope", "turn the rope"),
    ("fa", "get your ma"),
    ("fudge", "call the judge"),
    ("fat", "pet the cat"),
    ("fog", "walk the dog"),
    ("fun", "say we're done"),
    ]
start2 = "Someone better"
```

For each string pair ( `first`, `second` ) in `rhymes`:

For the first line:

- Print each string in `start1`, capitalized and followed by an exclamation point and a space.
- Print `first`, also capitalized and followed by an exclamation point.

For the second line:

- Print `start2` and a space.
- Print `second` and a period.

```
start1 = ["fee", "fie", "foe"]
rhymes = [
    ("flop", "get a mop"),
    ("fope", "turn the rope"),
    ("fa", "get your ma"),
    ("fudge", "call the judge"),
    ("fat", "pet the cat"),
    ("fog", "pet the dog"),
    ("fun", "say we're done"),
    ]
start2 = "Someone better"
start1_caps = " ".join([word.capitalize() + "!" for word in start1])
for first, second in rhymes:
    print(f"{start1_caps} {first.capitalize()}!")
    print(f"{start2} {second}.")
```

Output:

```
Fee! Fie! Foe! Flop!
Someone better get a mop.
Fee! Fie! Foe! Fope!
Someone better turn the rope.
Fee! Fie! Foe! Fa!
Someone better get your ma.
Fee! Fie! Foe! Fudge!
Someone better call the judge.
Fee! Fie! Foe! Fat!
Someone better pet the cat.
Fee! Fie! Foe! Fog!
Someone better walk the dog.
Fee! Fie! Foe! Fun!
Someone better say we're done.
```

---

# 8. Dictionaries

8.1 Make an English-to-French dictionary called `e2f` and print it. Here are your starter words: `dog` is `chien`, `cat` is `chat`, and `walrus` is

morse .

---

```
>>> e2f = {'dog': 'chien', 'cat': 'chat', 'walrus': 'morse'}
>>> e2f
{'cat': 'chat', 'walrus': 'morse', 'dog': 'chien'}
```

---

8.2 Using your three-word dictionary `e2f` , print the French word for `walrus` .

---

```
>>> e2f['walrus']
'morse'
```

---

8.3 Make a French-to-English dictionary called `f2e` from `e2f` . Use the `items` method.

---

```
>>> f2e = {}
>>> for english, french in e2f.items():
    f2e[french] = english
>>> f2e
{'morse': 'walrus', 'chien': 'dog', 'chat': 'cat'}
```

---

8.4 Print the English equivalent of the French word `chien` .

---

```
>>> f2e['chien']
'dog'
```

---

8.5 Print the set of English words from `e2f`.

---

```
>>> set(e2f.keys())
{'cat', 'walrus', 'dog'}
```

---

8.6 Make a multilevel dictionary called `life`. Use these strings for the topmost keys: `'animals'`, `'plants'`, and `'other'`. Make the `'animals'` key refer to another dictionary with the keys `'cats'`, `'octopi'`, and `'emus'`. Make the `'cats'` key refer to a list of strings with the values `'Henri'`, `'Grumpy'`, and `'Lucy'`. Make all the other keys refer to empty dictionaries.

---

This is a hard one, so don't feel bad if you peeked here first.

```
>>> life = {
...     'animals': {
...         'cats': [
...             'Henri', 'Grumpy', 'Lucy'
...             ],
...         'octopi': {},
...         'emus': {}
...         },
...     'plants': {},
...     'other': {}
...     }
>>>
```

---

8.7 Print the top-level keys of `life`.

```
>>> print(life.keys())
dict_keys(['animals', 'other', 'plants'])
```

Python 3 includes that `dict_keys` stuff. To print them as a plain list, use this:

```
>>> print(list(life.keys()))
['animals', 'other', 'plants']
```

By the way, you can use spaces to make your code easier to read:

```
>>> print ( list ( life.keys() ) )
['animals', 'other', 'plants']
```

---

8.8 Print the keys for `life['animals']`.

---

```
>>> print(life['animals'].keys())
dict_keys(['cats', 'octopi', 'emus'])
```

---

8.9 Print the values for `life['animals']['cats']`.

---

```
>>> print(life['animals']['cats'])
['Henri', 'Grumpy', 'Lucy']
```

---

8.10 Use a dictionary comprehension to create the dictionary `squares`. Use `range(10)` to return the keys, and use the square of each key as its value.

```
>>> squares = {key: key*key for key in range(10)}
>>> squares
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

---

8.11 Use a set comprehension to create the set `odd` from the odd numbers in `range(10)`.

---

```
>>> odd = {number for number in range(10) if number % 2 == 1}
>>> odd
{1, 3, 5, 7, 9}
```

---

8.12 Use a generator comprehension to return the string `'Got '` and a number for the numbers in `range(10)`. Iterate through this by using a `for` loop.

---

```
>>> for thing in ('Got %s' % number for number in range(10)):
...     print(thing)
...
Got 0
Got 1
Got 2
Got 3
Got 4
Got 5
Got 6
Got 7
Got 8
Got 9
```

---

8.13 Use `zip()` to make a dictionary from the key tuple `('optimist',`

'pessimist', 'troll') and the values tuple ('The glass is half full', 'The glass is half empty', 'How did you get a glass?').

---

```
>>> keys = ('optimist', 'pessimist', 'troll')
>>> values = ('The glass is half full',
...     'The glass is half empty',
...     'How did you get a glass?')
>>> dict(zip(keys, values))
{'optimist': 'The glass is half full',
'pessimist': 'The glass is half empty',
'troll': 'How did you get a glass?'}
```

---

8.14 Use `zip()` to make a dictionary called `movies` that pairs these lists: `titles = ['Creature of Habit', 'Crewel Fate', 'Sharks On a Plane']` and `plots = ['A nun turns into a monster', 'A haunted yarn shop', 'Check your exits']`

---

```
>>> titles = ['Creature of Habit',
...     'Crewel Fate',
...     'Sharks On a Plane']
>>> plots = ['A nun turns into a monster',
...     'A haunted yarn shop',
...     'Check your exits']
>>> movies = dict(zip(titles, plots))
>>> movies
{'Creature of Habit': 'A nun turns into a monster',
'Crewel Fate': 'A haunted yarn shop',
'Sharks On a Plane': 'Check your exits'}
>>>
```

---

# 9. Functions

9.1 Define a function called `good()` that returns the following list: `['Harry', 'Ron', 'Hermione']`.

---

```
>>> def good():
...     return ['Harry', 'Ron', 'Hermione']
...
>>> good()
['Harry', 'Ron', 'Hermione']
```

---

9.2 Define a generator function called `get_odds()` that returns the odd numbers from `range(10)`. Use a `for` loop to find and print the third value returned.

---

```
>>> def get_odds():
...     for number in range(1, 10, 2):
...         yield number
...
>>> count = 1
>>> for number in get_odds():
...     if count == 3:
...         print("The third odd number is", number)
...         break
...     count += 1
...
The third odd number is 5
```

---

9.3 Define a decorator called `test` that prints `'start'` when a function is called, and `'end'` when it finishes.

```
>>> def test(func):
...     def new_func(*args, **kwargs):
...         print('start')
...         result = func(*args, **kwargs)
...         print('end')
...         return result
...     return new_func
...
>>>
>>> @test
... def greeting():
...     print("Greetings, Earthling")
...
>>> greeting()
start
Greetings, Earthling
end
```

---

9.4 Define an exception called `OopsException`. Raise this exception to see what happens. Then, write the code to catch this exception and print `'Caught an oops'`.

```
>>> class OopsException(Exception):
...     pass
...
>>> raise OopsException()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.OopsException
>>>
>>> try:
...     raise OopsException
... except OopsException:
...     print('Caught an oops')
...
Caught an oops
```

---

# 10. Oh Oh: Objects and Classes

10.1 Make a class called `Thing` with no contents and print it. Then, create an object called `example` from this class and also print it. Are the printed values the same or different?

---

```
>>> class Thing:
...     pass
...
>>> print(Thing)
<class '__main__.Thing'>
>>> example = Thing()
>>> print(example)
<__main__.Thing object at 0x1006f3fd0>
```

---

10.2 Make a new class called `Thing2` and assign the value `'abc'` to a class variable called `letters`. Print `letters`.

```
>>> class Thing2:
...     letters = 'abc'
...
>>> print(Thing2.letters)
abc
```

---

10.3 Make yet another class called (of course) `Thing3`. This time, assign the value `'xyz'` to an instance (object) variable called `letters`. Print `letters`. Do you need to make an object from the class to do this?

---

```
>>> class Thing3:
...     def __init__(self):
...         self.letters = 'xyz'
...
```

The variable `letters` belongs to any objects made from `Thing3`, not the `Thing3` class itself:

```
>>> print(Thing3.letters)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'Thing3' has no attribute 'letters'
>>> something = Thing3()
>>> print(something.letters)
xyz
```

---

10.4 Make a class called `Element`, with instance attributes `name`, `symbol`, and `number`. Create an object called `hydrogen` of this class with the values `'Hydrogen'`, `'H'`, and `1`.

```
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.name = name
...         self.symbol = symbol
...         self.number = number
...
>>> hydrogen = Element('Hydrogen', 'H', 1)
```

---

10.5 Make a dictionary with these keys and values: `'name': 'Hydrogen', 'symbol': 'H', 'number': 1`. Then, create an object called `hydrogen` from class `Element` using this dictionary.

---

Start with the dictionary:

```
>>> el_dict = {'name': 'Hydrogen', 'symbol': 'H', 'number': 1}
```

This works, although it takes a bit of typing:

```
>>> hydrogen = Element(el_dict['name'], el_dict['symbol'], el_dict['number'])
```

Let's check that it worked:

```
>>> hydrogen.name
'Hydrogen'
```

However, you can also initialize the object directly from the dictionary, because its key names match the arguments to `__init__` (refer to Chapter 9 for a discussion of keyword arguments):

```
>>> hydrogen = Element(**el_dict)
>>> hydrogen.name
'Hydrogen'
```

---

10.6 For the `Element` class, define a method called `dump()` that prints the values of the object's attributes (`name`, `symbol`, and `number`). Create the `hydrogen` object from this new definition and use `dump()` to print its attributes.

---

```
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.name = name
...         self.symbol = symbol
...         self.number = number
...     def dump(self):
...         print('name=%s, symbol=%s, number=%s' %
...             (self.name, self.symbol, self.number))
...
>>> hydrogen = Element(**el_dict)
>>> hydrogen.dump()
name=Hydrogen, symbol=H, number=1
```

---

10.7 Call `print(hydrogen)`. In the definition of `Element`, change the name of the method `dump` to `__str__`, create a new `hydrogen` object, and call `print(hydrogen)` again.

```
>>> print(hydrogen)
<__main__.Element object at 0x1006f5310>
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.name = name
...         self.symbol = symbol
...         self.number = number
...     def __str__(self):
...         return ('name=%s, symbol=%s, number=%s' %
...             (self.name, self.symbol, self.number))
...
>>> hydrogen = Element(**el_dict)
>>> print(hydrogen)
name=Hydrogen, symbol=H, number=1
```

`__str__()` is one of Python's *magic methods*. The `print` function calls an object's `__str__()` method to get its string representation. If it doesn't have a `__str__()` method, it gets the default method from its parent `Object` class, which returns a string like `<__main__.Element object at 0x1006f5310>`.

---

10.8 Modify `Element` to make the attributes `name`, `symbol`, and `number` private. Define a getter property for each to return its value.

```
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.__name = name
...         self.__symbol = symbol
...         self.__number = number
...     @property
...     def name(self):
...         return self.__name
...     @property
...     def symbol(self):
...         return self.__symbol
...     @property
...     def number(self):
...         return self.__number
...
>>> hydrogen = Element('Hydrogen', 'H', 1)
>>> hydrogen.name
'Hydrogen'
>>> hydrogen.symbol
'H'
>>> hydrogen.number
1
```

---

10.9 Define three classes: `Bear`, `Rabbit`, and `Octothorpe`. For each, define only one method: `eats()`. This should return `'berries'` (`Bear`), `'clover'` (`Rabbit`), and `'campers'` (`Octothorpe`). Create one object from each and print what it eats.

```
>> class Bear:
...     def eats(self):
...         return 'berries'
...
>>> class Rabbit:
...     def eats(self):
...         return 'clover'
...
>>> class Octothorpe:
...     def eats(self):
...         return 'campers'
...
>>> b = Bear()
>>> r = Rabbit()
>>> o = Octothorpe()
>>> print(b.eats())
berries
>>> print(r.eats())
clover
>>> print(o.eats())
campers
```

---

10.10 Define these classes: `Laser`, `Claw`, and `SmartPhone`. Each has only one method: `does()`. This returns `'disintegrate'` (`Laser`), `'crush'` (`Claw`), or `'ring'` (`SmartPhone`). Then, define the class `Robot` that has one instance (object) of each of these. Define a `does()` method for the `Robot` that prints what its component objects do.

```
>>> class Laser:
...     def does(self):
...         return 'disintegrate'
...
>>> class Claw:
...     def does(self):
...         return 'crush'
...
>>> class SmartPhone:
...     def does(self):
...         return 'ring'
...
>>> class Robot:
...     def __init__(self):
...         self.laser = Laser()
...         self.claw = Claw()
...         self.smartphone = SmartPhone()
...     def does(self):
...         return '''I have many attachments:
... My laser, to %s.
... My claw, to %s.
... My smartphone, to %s.''' % (
...     self.laser.does(),
...     self.claw.does(),
...     self.smartphone.does() )
...
>>> robbie = Robot()
>>> print( robbie.does() )
I have many attachments:
My laser, to disintegrate.
My claw, to crush.
My smartphone, to ring.
```

# 11. Modules, Packages, and Goodies

11.1 Make a file called *zoo.py*. In it, define a function called `hours` that prints the string `'Open 9-5 daily'`. Then, use the interactive interpreter to import the `zoo` module and call its `hours` function.

Here's *zoo.py*:

```
def hours():
    print('Open 9-5 daily')
```

And now, let's import it interactively:

```
>>> import zoo
>>> zoo.hours()
Open 9-5 daily
```

---

11.2 In the interactive interpreter, import the `zoo` module as `menagerie` and call its `hours()` function.

---

```
>>> import zoo as menagerie
>>> menagerie.hours()
Open 9-5 daily
```

---

11.3 Staying in the interpreter, import the `hours()` function from `zoo` directly and call it.

---

```
>>> from zoo import hours
>>> hours()
Open 9-5 daily
```

---

11.4 Import the `hours()` function as `info` and call it.

```
>>> from zoo import hours as info
>>> info()
Open 9-5 daily
```

---

11.6 Make an `OrderedDict` called `fancy` from the same pairs and print it. Did it print in the same order as `plain`?

---

```
>>> from collections import OrderedDict
>>> fancy = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
>>> fancy
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

---

11.7 Make a `defaultdict` called `dict_of_lists` and pass it the argument `list`. Make the list `dict_of_lists['a']` and append the value `'something for a'` to it in one assignment. Print `dict_of_lists['a']`.

---

```
>>> from collections import defaultdict
>>> dict_of_lists = defaultdict(list)
>>> dict_of_lists['a'].append('something for a')
>>> dict_of_lists['a']
['something for a']
```

---

# 12. Wrangle and Mangle Data

12.1 Create a Unicode string called `mystery` and assign it the value `'\U0001f984'`. Print `mystery` and its Unicode name.

```
>>> import unicodedata
>>> mystery = '\U0001f4a9'
>>> mystery
'💩'
>>> unicodedata.name(mystery)
'PILE OF POO'
```

Oh my. What else have they got in there?

---

12.2 Encode `mystery`, this time using UTF-8, into the `bytes` variable `popbytes`. Print `pop_bytes`.

---

```
>>> pop_bytes = mystery.encode('utf-8')
>>> pop_bytes
b'\xf0\x9f\x92\xa9'
```

---

12.3 Using UTF-8, decode `popbytes` into the string variable `pop_string`. Print `pop_string`. Is `pop_string` equal to `mystery`?

---

```
>>> pop_string = pop_bytes.decode('utf-8')
>>> pop_string
'💩'
>>> pop_string == mystery
True
```

---

12.4 When you're working with text, regular expressions come in very handy. We'll apply them in a number of ways to our featured text sample. It's a poem titled "Ode on the Mammoth Cheese," written by James McIntyre in 1866 in homage to a seven-thousand-pound cheese that was

crafted in Ontario and sent on an international tour. If you'd rather not type all of it, use your favorite search engine and cut and paste the words into your Python program, or just grab it from Project Gutenberg. Call the text string `mammoth`.

---

```
>>> mammoth = '''
... We have seen thee, queen of cheese,
... Lying quietly at your ease,
... Gently fanned by evening breeze,
... Thy fair form no flies dare seize.
...
... All gaily dressed soon you'll go
... To the great Provincial show,
... To be admired by many a beau
... In the city of Toronto.
...
... Cows numerous as a swarm of bees,
... Or as the leaves upon the trees,
... It did require to make thee please,
... And stand unrivalled, queen of cheese.
...
... May you not receive a scar as
... We have heard that Mr. Harris
... Intends to send you off as far as
... The great world's show at Paris.
...
... Of the youth beware of these,
... For some of them might rudely squeeze
... And bite your cheek, then songs or glees
... We could not sing, oh! queen of cheese.
...
... We'rt thou suspended from balloon,
... You'd cast a shade even at noon,
... Folks would think it was the moon
... About to fall and crush them soon.
... '''
```

---

12.5 Import the `re` module to use Python's regular expression functions.

Use the `re.findall()` to print all the words that begin with `c`.

---

We'll define the variable `pat` for the pattern and then search for it in `mammoth`:

```
>>> import re
>>> pat = r'\bc\w*'
>>> re.findall(pat, mammoth)
['cheese', 'city', 'cheese', 'cheek', 'could', 'cheese', 'cast', 'crush']
```

The `\b` means to begin at a boundary between a word and a nonword. Use this to specify either the beginning or end of a word. The literal `c` is the first letter of the words we're looking for. The `\w` means any *word character*, which includes letters, digits, and underscores ( `_` ). The `*` means *zero or more* of these word characters. Together, this finds words that begin with `c`, including `'c'` itself. If you didn't use a raw string (with an `r` right before the starting quote), Python would interpret `\b` as a backspace and the search would mysteriously fail:

```
>>> pat = '\bc\w*'
>>> re.findall(pat, mammoth)
[]
```

---

12.6 Find all four-letter words that begin with `c`.

```
>>> pat = r'\bc\w{3}\b'
>>> re.findall(pat, mammoth)
['city', 'cast']
```

You need that final `\b` to indicate the end of the word. Otherwise, you'll get the first four letters of all words that begin with `c` and have at least four letters:

```
>>> pat = r'\bc\w{3}'
>>> re.findall(pat, mammoth)
['chee', 'city', 'chee', 'chee', 'coul', 'chee', 'cast', 'crus']
```

---

12.7 Find all the words that end with `r`.

This is a little tricky. We get the right result for words that end with `r`:

```
>>> pat = r'\b\w*r\b'
>>> re.findall(pat, mammoth)
['your', 'fair', 'Or', 'scar', 'Mr', 'far', 'For', 'your', 'or']
```

However, the results aren't so good for words that end with `l`:

```
>>> pat = r'\b\w*l\b'
>>> re.findall(pat, mammoth)
['All', 'll', 'Provincial', 'fall']
```

But what's that `ll` doing there? The `\w` pattern matches only letters, numbers, and underscores—not ASCII apostrophes. As a result, it grabs the final `ll` from `you'll`. We can handle this by adding an apostrophe to the set of characters to match. Our first try fails:

```
>>> pat = r'\b[\w']*l\b'
  File "<stdin>", line 1
    pat = r'\b[\w']*l\b'
```

Python points to the vicinity of the error, but it might take a while to see that the mistake was that the pattern string is surrounded by the same apostrophe/quote character. One way to solve this is to escape it with a backslash:

```
>>> pat = r'\b[\w\']*l\b'
>>> re.findall(pat, mammoth)
['All', "you'll", 'Provincial', 'fall']
```

Another way is to surround the pattern string with double quotes:

```
>>> pat = r"\b[\w']*l\b"
>>> re.findall(pat, mammoth)
['All', "you'll", 'Provincial', 'fall']
```

12.8 Find all the words that contain exactly three vowels in a row.

---

Begin with a word boundary, any number of *word* characters, three vowels, and then any nonvowel characters to the end of the word:

```
>>> pat = r'\b[^aeiou]*[aeiou]{3}[^aeiou]*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'beau\nIn', 'queen', 'squeeze', 'queen']
```

This looks right, except for that `'beau\nIn'` string. We searched `mammoth` as a single multiline string. Our `[^aeiou]` matches any nonvowels, including `\n` (line feed, which marks the end of a text line). We need to add one more thing to the ignore set: `\s` matches any space characters, including `\n`:

```
>>> pat = r'\b\w*[aeiou]{3}[^aeiou\s]\w*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'queen', 'squeeze', 'queen']
```

We didn't find `beau` this time, so we need one more tweak to the pattern: match any number (even zero) of nonvowels after the three vowels. Our previous pattern always matched one nonvowel.

```
>>> pat = r'\b\w*[aeiou]{3}[^aeiou\s]*\w*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'beau', 'queen', 'squeeze', 'queen']
```

What does all of this show? Among other things, that regular expressions can do a lot, but they can be very tricky to get right.

---

12.9 Use `unhexlify()` to convert this hex string (combined from two strings to fit on a page) to a `bytes` variable called `gif`:

```
'47494638396101000100800000000000ffffff21f9' +
```

```
'0401000000002c0000000001000100000020144003b'
```

---

```
>>> import binascii
>>> hex_str = '47494638396101000100800000000000ffffff21f9' + \
...     '0401000000002c0000000001000100000020144003b'
>>> gif = binascii.unhexlify(hex_str)
>>> len(gif)
42
```

---

12.10 The bytes in `gif` define a one-pixel transparent GIF file, one of the most common graphics file formats. A legal GIF starts with the string *GIF89a*. Does `gif` match this?

---

```
>>> gif[:6] == b'GIF89a'
True
```

Notice that we needed to use a `b` to define a byte string rather than a Unicode character string. You can compare bytes with bytes, but you cannot compare bytes with strings:

```
>>> gif[:6] == 'GIF89a'
False
>>> type(gif)
<class 'bytes'>
>>> type('GIF89a')
<class 'str'>
>>> type(b'GIF89a')
<class 'bytes'>
```

---

12.11 The pixel width of a GIF is a 16-bit little-endian integer starting at byte offset 6, and the height is the same size, starting at offset 8. Extract

and print these values for `gif`. Are they both `1`?

---

```
>>> import struct
>>> width, height = struct.unpack('<HH', gif[6:10])
>>> width, height
(1, 1)
```

---

# 13. Calendars and Clocks

13.1 Write the current date as a string to the text file *today.txt*.

---

```
>>> from datetime import date
>>> now = date.today()
>>> now_str = now.isoformat()
>>> with open('today.txt', 'wt') as output:
...     print(now_str, file=output)
>>>
```

When I ran this, here's what I got in *today.txt*:

```
2019-07-23
```

Instead of `print`, you could have also said something like `output.write(now_str)`. Using `print` adds the final newline.

---

13.2 Read the text file *today.txt* into the string `today_string`.

```
>>> with open('today.txt', 'rt') as input:
...     today_string = input.read()
...
>>> today_string
'2019-07-23\n'
```

---

13.3 Parse the date from `today_string`.

---

```
>>> fmt = '%Y-%m-%d\n'
>>> datetime.strptime(today_string, fmt)
datetime.datetime(2019, 7, 23, 0, 0)
```

If you wrote that final newline to the file, you need to match it in the format string.

---

13.4 Create a date object of your day of birth.

---

Let's say that you were born on August 14, 1982:

```
>>> my_day = date(1982, 8, 14)
>>> my_day
datetime.date(1982, 8, 14)
```

---

13.5 What day of the week was your day of birth?

```
>>> my_day.weekday()
5
>>> my_day.isoweekday()
6
```

With `weekday()`, Monday is 0 and Sunday is 6. With `isoweekday()`, Monday is 1 and Sunday is 7. Therefore, this date was a Saturday.

---

13.6 When will you be (or when were you) 10,000 days old?

---

```
>>> from datetime import timedelta
>>> party_day = my_day + timedelta(days=10000)
>>> party_day
datetime.date(2009, 12, 30)
```

If August 14, 1982 was your birthday, you probably missed an excuse for a party.

---

# 14. Files and Directories

14.1 List the files in your current directory.

---

If your current directory is *ohmy* and contains three files named after animals, it might look like this:

```
>>> import os
>>> os.listdir('.')
['bears', 'lions', 'tigers']
```

---

14.2 List the files in your parent directory.

---

If your parent directory contained two files plus the current *ohmy* directory, it might look like this:

```
>>> import os
>>> os.listdir('..')
['ohmy', 'paws', 'whiskers']
```

---

14.3 Assign the string `'This is a test of the emergency text system'` to the variable `test1`, nd write `test1` to a file called *test.txt*.

---

```
>>> test1 = 'This is a test of the emergency text system'
>>> len(test1)
43
```

Here's how to do it by using `open`, `write`, and `close`:

```
>>> outfile = open('test.txt', 'wt')
>>> outfile.write(test1)
43
>>> outfile.close()
```

Or, you can use `with` and avoid calling `close` (Python does it for you):

```
>>> with open('test.txt', 'wt') as outfile:
...     outfile.write(test1)
...
43
```

---

14.4 Open the file *test.txt* and read its contents into the string `test2`. Are

`test1` and `test2` the same?

---

```
>>> with open('test.txt', 'rt') as infile:
...     test2 = infile.read()
...
>>> len(test2)
43
>>> test1 == test2
True
```

---

# 15. Data in Time: Processes and Concurrency

15.1 Use `multiprocessing` to create three separate processes. Make each one wait a random number of seconds between zero and one, print the current time, and then exit.

```
import multiprocessing

def now(seconds):
    from datetime import datetime
    from time import sleep
    sleep(seconds)
    print('wait', seconds, 'seconds, time is', datetime.utcnow())

if __name__ == '__main__':
    import random
    for n in range(3):
        seconds = random.random()
        proc = multiprocessing.Process(target=now, args=(seconds,))
        proc.start()
```

```
$ python multi_times.py
wait 0.10720361113059229 seconds, time is 2019-07-24 00:19:23.951594
wait 0.5825144002370065 seconds, time is 2019-07-24 00:19:24.425047
wait 0.6647690569029477 seconds, time is 2019-07-24 00:19:24.509995
```

---

# 16. Data in a Box: Persistent Storage

16.1 Save the following text lines to a file called *books.csv* (notice that if the fields are separated by commas, you need to surround a field with quotes if it contains a comma):

```
author,book
J R R Tolkien,The Hobbit
Lynne Truss,"Eats, Shoots & Leaves"
```

```
>>> text = '''author,book
... J R R Tolkien,The Hobbit
... Lynne Truss,"Eats, Shoots & Leaves"
... '''
>>> with open('test.csv', 'wt') as outfile:
...     outfile.write(text)
...
73
```

---

16.2 Use the `csv` module and its `DictReader` method to read *books.csv* to the variable `books`. Print the values in `books`. Did `DictReader` handle the quotes and commas in the second book's title?

---

```
>>> with open('books.csv', 'rt') as infile:
...     books = csv.DictReader(infile)
...     for book in books:
...         print(book)
...
{'book': 'The Hobbit', 'author': 'J R R Tolkien'}
{'book': 'Eats, Shoots & Leaves', 'author': 'Lynne Truss'}
```

---

16.3 Create a CSV file called *books2.csv* by using these lines:

```
title,author,year
The Weirdstone of Brisingamen,Alan Garner,1960
Perdido Street Station,China Miéville,2000
Thud!,Terry Pratchett,2005
The Spellman Files,Lisa Lutz,2007
Small Gods,Terry Pratchett,1992
```

```
>>> text = '''title,author,year
... The Weirdstone of Brisingamen,Alan Garner,1960
... Perdido Street Station,China Miéville,2000
... Thud!,Terry Pratchett,2005
... The Spellman Files,Lisa Lutz,2007
... Small Gods,Terry Pratchett,1992
... '''
>>> with open('books2.csv', 'wt') as outfile:
...     outfile.write(text)
...
201
```

---

16.4 Use the `sqlite3` module to create a SQLite database called *books.db* and a table called `books` with these fields: `title` (text), `author` (text), and `year` (integer).

---

```
>>> import sqlite3
>>> db = sqlite3.connect('books.db')
>>> curs = db.cursor()
>>> curs.execute('''create table book (title text, author text, year int)''')
<sqlite3.Cursor object at 0x1006e3b90>
>>> db.commit()
```

---

16.5 Read *books2.csv* and insert its data into the `book` table.

```
>>> import csv
>>> import sqlite3
>>> ins_str = 'insert into book values(?, ?, ?)'
>>> with open('books.csv', 'rt') as infile:
...   books = csv.DictReader(infile)
...   for book in books:
...     curs.execute(ins_str, (book['title'], book['author'], book['year']))
...
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
>>> db.commit()
```

16.6 Select and print the `title` column from the `book` table in alphabetical order.

```
>>> sql = 'select title from book order by title asc'
>>> for row in db.execute(sql):
...     print(row)
...
('Perdido Street Station',)
('Small Gods',)
('The Spellman Files',)
('The Weirdstone of Brisingamen',)
('Thud!',)
```

If you just wanted to print the `title` value without that tuple stuff (parentheses and comma), try this:

```
>>> for row in db.execute(sql):
...     print(row[0])
...
Perdido Street Station
Small Gods
The Spellman Files
The Weirdstone of Brisingamen
Thud!
```

If you want to ignore the initial `'The'` in titles, you need a little extra SQL fairy dust:

```
>>> sql = '''select title from book order by
... case when (title like "The %")
... then substr(title, 5)
... else title end'''
>>> for row in db.execute(sql):
...     print(row[0])
...
Perdido Street Station
Small Gods
The Spellman Files
Thud!
The Weirdstone of Brisingamen
```

16.7 Select and print all columns from the `book` table in order of publication.

---

```
>>> for row in db.execute('select * from book order by year'):
...     print(row)
...
('The Weirdstone of Brisingamen', 'Alan Garner', 1960)
('Small Gods', 'Terry Pratchett', 1992)
('Perdido Street Station', 'China Miéville', 2000)
('Thud!', 'Terry Pratchett', 2005)
('The Spellman Files', 'Lisa Lutz', 2007)
```

To print all the fields in each row, just separate with a comma and space:

```
>>> for row in db.execute('select * from book order by year'):
...     print(*row, sep=', ')
...
The Weirdstone of Brisingamen, Alan Garner, 1960
Small Gods, Terry Pratchett, 1992
Perdido Street Station, China Miéville, 2000
Thud!, Terry Pratchett, 2005
The Spellman Files, Lisa Lutz, 2007
```

---

16.8 Use the `sqlalchemy` module to connect to the sqlite3 database *books.db* that you just made in exercise 8.6. As in 8.8, select and print the `title` column from the `book` table in alphabetical order.

```
>>> import sqlalchemy
>>> conn = sqlalchemy.create_engine('sqlite:///books.db')
>>> sql = 'select title from book order by title asc'
>>> rows = conn.execute(sql)
>>> for row in rows:
...     print(row)
...
('Perdido Street Station',)
('Small Gods',)
('The Spellman Files',)
('The Weirdstone of Brisingamen',)
('Thud!',)
```

---

16.9 Install the Redis server (see Appendix B) and the Python `redis` library (`pip install redis`) on your machine. Create a Redis hash called `test` with the fields `count` (`1`) and `name` (`'Fester Bestertester'`). Print all the fields for `test`.

---

```
>>> import redis
>>> conn = redis.Redis()
>>> conn.delete('test')
1
>>> conn.hmset('test', {'count': 1, 'name': 'Fester Bestertester'})
True
>>> conn.hgetall('test')
{b'name': b'Fester Bestertester', b'count': b'1'}
```

---

16.10 Increment the `count` field of `test` and print it.

```
>>> conn.hincrby('test', 'count', 3)
4
>>> conn.hget('test', 'count')
b'4'
```

---

# 17. Data in Space: Networks

17.1 Use a plain `socket` to implement a current-time service. When a client sends the string `'time'` to the server, return the current date and time as an ISO string.

Here's one way to write the server, *udp_time_server.py*:

```
from datetime import datetime
import socket

address = ('localhost', 6789)
max_size = 4096

print('Starting the server at', datetime.now())
print('Waiting for a client to call.')
server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(address)
while True:
    data, client_addr = server.recvfrom(max_size)
    if data == b'time':
        now = str(datetime.utcnow())
        data = now.encode('utf-8')
        server.sendto(data, client_addr)
        print('Server sent', data)
server.close()
```

And the client, *udp_time_client.py*:

```
import socket
from datetime import datetime
from time import sleep

address    = ('localhost', 6789)
max_size   = 4096

print('Starting the client at', datetime.now())
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
    sleep(5)
    client.sendto(b'time', address)
    data, server_addr = client.recvfrom(max_size)
    print('Client read', data)
client.close()
```

I put in a `sleep(5)` call at the top of the client loop to make the data ex-

change less supersonic. Start the server in one window:

```
$ python udp_time_server.py
Starting the server at 2014-06-02 20:28:47.415176
Waiting for a client to call.
```

Start the client in another window:

```
$ python udp_time_client.py
Starting the client at 2014-06-02 20:28:51.454805
```

After five seconds, you'll start getting output in both windows. Here are the first three lines from the server:

```
Server sent b'2014-06-03 01:28:56.462565'
Server sent b'2014-06-03 01:29:01.463906'
Server sent b'2014-06-03 01:29:06.465802'
```

And here are the first three from the client:

```
Client read b'2014-06-03 01:28:56.462565'
Client read b'2014-06-03 01:29:01.463906'
Client read b'2014-06-03 01:29:06.465802'
```

Both of these programs run forever, so you'll need to cancel them manually.

---

17.2. Use ZeroMQ REQ and REP sockets to do the same thing.

```
import zmq
from datetime import datetime

host = '127.0.0.1'
port = 6789
context = zmq.Context()
server = context.socket(zmq.REP)
server.bind("tcp://%s:%s" % (host, port))
print('Server started at', datetime.utcnow())
while True:
    #  Wait for next request from client
    message = server.recv()
    if message == b'time':
        now = datetime.utcnow()
        reply = str(now)
        server.send(bytes(reply, 'utf-8'))
        print('Server sent', reply)
```

```
import zmq
from datetime import datetime
from time import sleep

host = '127.0.0.1'
port = 6789
context = zmq.Context()
client = context.socket(zmq.REQ)
client.connect("tcp://%s:%s" % (host, port))
print('Client started at', datetime.utcnow())
while True:
    sleep(5)
    request = b'time'
    client.send(request)
    reply = client.recv()
    print("Client received %s" % reply)
```

With plain sockets, you need to start the server first. With ZeroMQ, you can start either the server or client first.

```
$ python zmq_time_server.py
```

```
Server started at 2014-06-03 01:39:36.933532
```

```
$ python zmq_time_client.py
Client started at 2014-06-03 01:39:42.538245
```

After 15 seconds or so, you should have some lines from the server:

```
Server sent 2014-06-03 01:39:47.539878
Server sent 2014-06-03 01:39:52.540659
Server sent 2014-06-03 01:39:57.541403
```

Here's what you should see from the client:

```
Client received b'2014-06-03 01:39:47.539878'
Client received b'2014-06-03 01:39:52.540659'
Client received b'2014-06-03 01:39:57.541403'
```

---

17.3. Try the same with XMLRPC.

From the server:

```
from xmlrpc.server import SimpleXMLRPCServer

def now():
    from datetime import datetime
    data = str(datetime.utcnow())
    print('Server sent', data)
    return data

server = SimpleXMLRPCServer(("localhost", 6789))
server.register_function(now, "now")
server.serve_forever()
```

And from the client:

```
import xmlrpc.client
from time import sleep

proxy = xmlrpc.client.ServerProxy("http://localhost:6789/")
while True:
    sleep(5)
    data = proxy.now()
    print('Client received', data)
```

Start the server:

```
$ python xmlrpc_time_server.py
```

Start the client:

```
$ python xmlrpc_time_client.py
```

Wait 15 seconds or so. Here are the first three lines of server output:

```
Server sent 2014-06-03 02:14:52.299122
127.0.0.1 - - [02/Jun/2014 21:14:52] "POST / HTTP/1.1" 200 -
```

```
Server sent 2014-06-03 02:14:57.304741
127.0.0.1 - - [02/Jun/2014 21:14:57] "POST / HTTP/1.1" 200 -
Server sent 2014-06-03 02:15:02.310377
127.0.0.1 - - [02/Jun/2014 21:15:02] "POST / HTTP/1.1" 200 -
```

And here are the first three lines from the client:

```
Client received 2014-06-03 02:14:52.299122
Client received 2014-06-03 02:14:57.304741
Client received 2014-06-03 02:15:02.310377
```

---

17.4 You may have seen the classic *I Love Lucy* television episode in which Lucy and Ethel worked in a chocolate factory. The duo fell behind as the conveyor belt that supplied the confections for them to process began operating at an ever-faster rate. Write a simulation that pushes different types of chocolates to a Redis list, and Lucy is a client doing blocking pops of this list. She needs 0.5 seconds to handle a piece of chocolate. Print the time and type of each chocolate as Lucy gets it, and how many remain to be handled.

*redis_choc_supply.py* supplies the infinite treats:

```
import redis
import random
from time import sleep

conn = redis.Redis()
varieties = ['truffle', 'cherry', 'caramel', 'nougat']
conveyor = 'chocolates'
while True:
    seconds = random.random()
    sleep(seconds)
    piece = random.choice(varieties)
    conn.rpush(conveyor, piece)
```

*redis_lucy.py* might look like this:

```
import redis
from datetime import datetime
from time import sleep

conn = redis.Redis()
timeout = 10
conveyor = 'chocolates'
while True:
    sleep(0.5)
    msg = conn.blpop(conveyor, timeout)
    remaining = conn.llen(conveyor)
    if msg:
        piece = msg[1]
        print('Lucy got a', piece, 'at', datetime.utcnow(),
        ', only', remaining, 'left')
```

Start them in either order. Because Lucy takes a half second to handle each, and they're being produced every half second on average, it's a race to keep up. The more of a head start that you give to the conveyor belt, the harder you make Lucy's life.

```
$ python redis_choc_supply.py &


$ python redis_lucy.py
Lucy got a b'nougat' at 2014-06-03 03:15:08.721169 , only 4 left
Lucy got a b'cherry' at 2014-06-03 03:15:09.222816 , only 3 left
Lucy got a b'truffle' at 2014-06-03 03:15:09.723691 , only 5 left
Lucy got a b'truffle' at 2014-06-03 03:15:10.225008 , only 4 left
Lucy got a b'cherry' at 2014-06-03 03:15:10.727107 , only 4 left
Lucy got a b'cherry' at 2014-06-03 03:15:11.228226 , only 5 left
Lucy got a b'cherry' at 2014-06-03 03:15:11.729735 , only 4 left
Lucy got a b'truffle' at 2014-06-03 03:15:12.230894 , only 6 left
Lucy got a b'caramel' at 2014-06-03 03:15:12.732777 , only 7 left
Lucy got a b'cherry' at 2014-06-03 03:15:13.234785 , only 6 left
Lucy got a b'cherry' at 2014-06-03 03:15:13.736103 , only 7 left
Lucy got a b'caramel' at 2014-06-03 03:15:14.238152 , only 9 left
Lucy got a b'cherry' at 2014-06-03 03:15:14.739561 , only 8 left
```

Poor Lucy.

17.5 Use ZeroMQ to pub- lish the poem from ex- er- cise 12.4 (from Example 12-1), one

word at a time. Write a ZeroMQ con- sumer that prints ev- ery word that starts with a vowel, and an- other that prints ev- ery word that con- tains five let- ters. Ignore punc- tu- a-

tion
char-
ac-
ters.

Here's the server, *poem_pub.py*, which plucks each word from the poem and pub- lishes it to the topic `v o w e l s` if it starts with a vowel, and the topic `f i v`

e
if
it
has
five
let-
ters.
Some
words
might
be
in
both
top-
ics,
some
in
nei-
ther.

```
import string
import zmq

host = '127.0.0.1'
port = 6789
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
pub.bind('tcp://%s:%s' % (host, port))

with open('mammoth.txt', 'rt') as poem:
    words = poem.read()
for word in words.split():
    word = word.strip(string.punctuation)
    data = word.encode('utf-8')
    if word.startswith(('a','e','i','o','u','A','e','i','o','u')):
        pub.send_multipart([b'vowels', data])
    if len(word) == 5:
        pub.send_multipart([b'five', data])
```

The client, *poem_sub.py*, subscribes to the topics v o w e l s and f i v e and prints the topic and word:

```
import string
import zmq

host = '127.0.0.1'
port = 6789
ctx = zmq.Context()
sub = ctx.socket(zmq.SUB)
sub.connect('tcp://%s:%s' % (host, port))
sub.setsockopt(zmq.SUBSCRIBE, b'vowels')
sub.setsockopt(zmq.SUBSCRIBE, b'five')
while True:
```

```
        topic, word = sub.recv_multipart()
        print(topic, word)
```

If you start these and run them, they *almost* work. Your code looks fine but nothing happens. You need to read the ZeroMQ guide to learn about the *slow joiner* prob-

lem: even if you start the client be- fore the server, the server be- gins push- ing data im- me- di- ately af- ter start- ing, and the client takes a lit- tle time to con- nect

to the server. If you're publishing a constant stream of something and don't really care when the subscribers jump in, it's no problem. But in this case, the data stream

is
so
short
that
it’s
flown
past
be-
fore
the
sub-
scriber
blinks,
like
a
fast-
ball
past
a
bat-
ter.

The
eas-
i-
est
way
to
fix
this
is
to
make
the
pub-
lisher
sleep

a second after it calls `bind()` and before it starts sending messages. Call this version *poem_pub_sleep.py*:

```
import string
import zmq
from time import sleep

host = '127.0.0.1'
port = 6789
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
pub.bind('tcp://%s:%s' % (host, port))
```

```
    sleep(1)

    with open('mammoth.txt', 'rt') as poem:
        words = poem.read()
    for word in words.split():
        word = word.strip(string.punctuation)
        data = word.encode('utf-8')
        if word.startswith(('a','e','i','o','u','A','e','i','o','u')):
            print('vowels', data)
            pub.send_multipart([b'vowels', data])
        if len(word) == 5:
            print('five', data)
            pub.send_multipart([b'five', data])
```

Start the sub-scriber and then the sleepy pub-lisher:

```
$ python poem_sub.py
```

```
$ python poem_pub_sleep.py
```

Now, the sub-scriber has time to grab

its two topics. Here are the first few lines of its output:

```
b'five' b'queen'
b'vowels' b'of'
b'five' b'Lying'
b'vowels' b'at'
b'vowels' b'ease'
b'vowels' b'evening'
b'five' b'flies'
b'five' b'seize'
b'vowels' b'All'
b'five' b'gaily'
b'five' b'great'
b'vowels' b'admired'
```

If you can’t add a

```
s
l
e
e
p
```

(
)
to
your
pub-
lisher,
you
can
syn-
chro-
nize
pub-
lisher
and
sub-
scriber
pro-
grams
by
us-
ing
`R`
`E`
`Q`
and
`R`
`E`
`P`
sock-
ets.
See
the
*pub-*
*lisher.py*
and
*sub-*
*scriber.py*

ex-
am-
ples
on
GitHub.

# 18. The Web, Untangle

# d

18.1 If you haven’t installed `flask` yet, do so now. This will also install `werkzeug`, `jinja2`,

and
pos-
si-
bly
other
pack-
ages.

18.2
Build
a
skele-
ton
web-
site,
us-
ing
Flask’s
de-
bug/reload
de-
vel-
op-
ment
web
server.
Ensure
that
the
server
starts
up
for
host-
name
1
o

c
a
l
h
o
s
t
on
de-
fault
port
5
0
0
0
.
If
your
ma-
chine
is
al-
ready
us-
ing
port
5000
for
some-
thing
else,
use
an-
other
port
num-
ber.

Here's *flask1.py*:

```
from flask import Flask

app = Flask(__name__)

app.run(port=5000, debug=True)
```

Gentlemen, start your engines:

```
$ python flask1.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

18.3 Add a `home()` function to handle requests

for the home page. Set it up to re-turn the string `It's alive!`.

What should we call this one, *flask2.py*?

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "It's alive!"

app.run(debug=True)
```

Start the server:

```
$ python flask2.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

Finally, access the home page via a browser, command-

line HTTP program such as `curl` or `wget`, or even `telnet`:

```
$ curl http://localhost:5000/
It's alive!
```

18.4 Create a Jinja2 template file called

*home.html* with the following contents:

```
I'm of course referring to {{thing}},
which is {{height}} feet tall and {{color}}.
```

Make a directory called *templates* and create the file *home.html* with the contents just shown. If your Flask server is

still running from the previous examples, it will detect the new content and restart itself.

18.5 Modify your server's `home()` function to

use the *home.html* template. Provide it with three `GET` parameters: `thing`, `height`, and `color`.

Here comes *flask3.py*:

```
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def home():
    thing = request.values.get('thing')
    height = request.values.get('height')
    color = request.values.get('color')
    return render_template('home.html',
        thing=thing, height=height, color=color)

app.run(debug=True)
```

Go to this address in your web client:

```
http://localhost:5000/?thing=Octothorpe&height=7&color=green
```

You should see the following:

I'm of course referring to Octothorpe, which is 7 feet tall and green.

# 19. Beapythonista

(Pythonistas don't have home-work to-

day.)

# 20. PyArt

20.1 Install `matplotlib`. Draw a scatter diagram of these

(x,
y)
pairs:

```
 (
(
0
,
0
)
,
(
3
,
5
)
,
(
6
,
2
)
,
(
9
,
8
)
,
(
1
4
,
1
0
)
).
```

20.2 Draw a line graph of the same data.

20.3 Draw a plot (a line graph with mark-ers) of the same data

This has all three as sub-plots:

```
import matplotlib.pyplot as plt

x = (0, 3, 6, 9, 14)
y = (0, 5, 2, 8, 10)
fig, plots = plt.subplots(nrows=1, ncols=3)

plots[0].scatter(x, y)
plots[1].plot(x, y)
plots[2].plot(x, y, 'o-')

plt.show()
```

# 21. Python Tworo

k

21.1 Install geopandas and run Example 21-1. Try modifying things like colors and marker sizes.

2

2

.

P

y

S

c

i

22.1 Install Pandas. Get the CSV file in Example 16-1. Run the program in Example 16-2. Experiment with some of the Pandas commands.