Table of Contents

The Python Challenge Solutions

Spoiler Alert!

If you haven't heard "The Python Challenge", try it out now! I can wait.

Before you flip to the next page, be careful that this eBook contains solutions to ALL the challenges, which may spoil your journey. If you do not care about winning or losing, if you have no intention to brag about your "achievements" by cheating with these solutions, but simply want to learn Python in a fun and addictive way, feel free to move forward.

By the way, do me a favor, let's use Python 3! It is time to move on!

Learning Python

This eBook provides "just enough" knowledge for solving each challenge. To learn more about Python, try these resources:

- The Official Documentation: not always friendly for beginners, but it should be treated as the single source of truth.
- Books: books can be slightly outdated, however they may provide better entry level tutorials, or dig deeper in a specific topic
    - Basics: Learning Python
    - More Advanced: Effective Python
    - Machine Learning: Python Machine Learning
    - Data Analysis: Python for Data Analysis
    - Data Visualization: Data Visualization with Python and JavaScript
    - Hacking: Black Hat Python
    - Web Scraping: Web Scraping with Python
- HackingNote: last but not least, a shameless plug-in. I'm still learning Python, and I'm taking notes and shaping my knowledge base along the way. Hope it could help you as well.

One More Word...

I'm not a Python expert. I'm not an editor, not even a native English speaker. Please bear with me if I make any grammar or technical mistakes. Feel free to leave comments at the bottom of each page(if you are reading from the website).

Python Challenge - Level 0

- Link: http://www.pythonchallenge.com/pc/def/0.html

Problem

> Hint: try to change the URL address.

Explore

As the "0" indicates, this is just a warm up.

Loop

The naive way to solve it: multiply by 2 in a loop:

```
>>> k = 1
>>> for i in range(38):
...     k *= 2
...
>>> k
274877906944
```

Power

Use `**` for power:

```
>>> 2**38
274877906944
```

In REPL the result will be printed; or you can explicitly print the result by calling `print()`

```
>>> print(2**38)
274877906944
```

Instead of `**`, you can also use `pow()`

```
>>> pow(2,38)
274877906944
```

or

```
>>> print(pow(2,38))
274877906944
```

Shift

Multiply by 2 is equivalent to shifting the binary representation left by one:

```
>>> 1<<38
274877906944
```

Numeric Types

Done!

Wait, why 38? what is implied?

If you are coming from C/C++, Java or other languages, you know that there are multiple types just for integers: `short`, `integer`, `long`, and even `BigInteger` beyond 64-bit. However that is not the case in Python:

```
>>> type(2**3)
<class 'int'>
>>> type(2**38)
<class 'int'>
>>> type(2**380)
<class 'int'>
```

So 38 is a good(and random) example to show a `int` larger than 32-bit.

Similar for float type, in Python it can be arbitrarily large, no `double` needed

```
>>> type(2**3.8)
<class 'float'>
>>> type(2.0**38)
<class 'float'>
```

Complete Solution

```
print(2**38)
```

Output:

```
274877906944
```

Next Level

http://www.pythonchallenge.com/pc/def/274877906944.html

And it will jump to

http://www.pythonchallenge.com/pc/def/map.html

Read More

- Python 3 - REPL

Python Challenge - Level 1

- Link: http://www.pythonchallenge.com/pc/def/map.html

Problem

The image lists 3 pairs:

- K->M
- O->Q
- E->G

> everybody thinks twice before solving this.
>
> g fmnc wms bgblr rpylqjyrc gr zw fylb. rfyrq ufyr amknsrcpq ypc dmp. bmgle gr gl zw fylb gq glcddgagclr ylb rfyr'q ufw rfgq rcvr gq qm jmle. sqgle qrpgle.kyicrpylq() gq pcamkkclbcb. lmu ynnjw ml rfc spj.

Hints

The text must be encoded.

Hint from url: map. And notice that there's exactly one character between the given pairs: `K->L->M`, `O->P->Q`, `E->F->G`, and look at the very first letter in the text `g`, if this is English, a good guess is that `g` is actually `i`. What is the distance from `g` to `i`? Yes, `G->H->I`. So let's shift each character to the right, by 2, and `y` back to `a`, `z` back to `b`.

The string is not so long, so let's just copy and past it to REPL:

```
>>> raw = "g fmnc wms bgblr rpylqjyrc gr zw fylb. rfyrq ufyr amk
nsrcpq ypc dmp. bmgle grgl zw fylb gq glcddgagclr ylb rfyr'q ufw
 rfgq rcvr gq qm jmle. sqgle qrpgle.kyicrpylq() gq pcamkkclbcb.
lmu ynnjw ml rfc spj."
```

Solution 1

The pair of functions that would help up make the change:

- `ord()`: character to integer
- `chr()`: integer to character

e.g.

```
>>> chr(65)
'A'
>>> ord('A')
65
```

To shift a character:

```
>>> ord('k')
107
>>> ord('k') + 2
109
>>> chr(ord('k') + 2)
'm'
```

but it is not working for 'z':

```
>>> chr(ord('z') + 2)
'|'
```

To make it circular, calculate it's distance from 'a'

```
>>> (ord('z') + 2) - ord('a')
27
```

if it is larger than 26, go back to the beginning

```
>>> ((ord('z') + 2) - ord('a')) % 26
1
```

then add that difference to `'a'`

```
>>> chr(((ord('z') + 2) - ord('a')) % 26 + ord('a'))
'b'
```

Let's translate the whole string:

```
>>> result = ""
>>> for c in raw:
...     if c >= 'a' and c <= 'z':
...         result += chr(((ord(c) + 2) - ord('a')) % 26 + ord('
a'))
...     else:
...         result += c
...
>>> result
"i hope you didnt translate it by hand. thats what computers are
 for. doing itin by hand is inefficient and that's why this text
 is so long. using string.maketrans() is recommended. now apply
on the url."
```

That is how we make a loop in Java or C, but python has a better way: list comprehension

```
>>> ''.join([chr(((ord(s) + 2) - ord('a')) % 26 + ord('a')) if s
 >= 'a' and s <= 'z' else s for s in raw])
"i hope you didnt translate it by hand. thats what computers are
 for. doing itin by hand is inefficient and that's why this text
 is so long. using string.maketrans() is recommended. now apply
on the url."
```

But since it suggest us use `.maketrans()`, let's try it next.

Put Everything Together

```
raw = "g fmnc wms bgblr rpylqjyrc gr zw fylb. rfyrq ufyr amknsrc
pq ypc dmp. bmgle grgl zw fylb gq glcddgagclr ylb rfyr'q ufw rfg
q rcvr gq qm jmle. sqgle qrpgle.kyicrpylq() gq pcamkkclbcb. lmu
ynnjw ml rfc spj."

print(''.join([chr(((ord(s) + 2) - ord('a')) % 26 + ord('a')) if
 s >= 'a' and s <= 'z' else s for s in raw]))
```

Solution 2: .maketrans()

In Python 3, `.maketrans` is not in `string` as indicated; instead call `str.maketrans()` or `bytes.maketrans()`

```
>>> table = str.maketrans("abcdefghijklmnopqrstuvwxyz", "cdefghi
jklmnopqrstuvwxyzab")
>>> raw.translate(table)
"i hope you didnt translate it by hand. thats what computers are
 for. doing itin by hand is inefficient and that's why this text
 is so long. using string.maketrans() is recommended. now apply
on the url."
```

Put Everything Together

```
raw = "g fmnc wms bgblr rpylqjyrc gr zw fylb. rfyrq ufyr amknsrc
pq ypc dmp. bmgle grgl zw fylb gq glcddgagclr ylb rfyr'q ufw rfg
q rcvr gq qm jmle. sqgle qrpgle.kyicrpylq() gq pcamkkclbcb. lmu
ynnjw ml rfc spj."

table = str.maketrans(
    "abcdefghijklmnopqrstuvwxyz", "cdefghijklmnopqrstuvwxyzab"
)

result = raw.translate(table)

print(result)
```

Solution 3

Let's have our own version of maketrans. The inputs are two lists, then each character is mapped from one list to another.

Define the two lists

```
>>> a = "abcdefghijklmnopqrstuvwxyz,. '()"
>>> b = "cdefghijklmnopqrstuvwxyzab,. '()"
```

zip them

```
>>> list(zip(a, b))
[('a', 'c'), ('b', 'd'), ('c', 'e'), ('d', 'f'), ('e', 'g'), ('f'
, 'h'), ('g', 'i'), ('h', 'j'), ('i', 'k'), ('j', 'l'), ('k', 'm'
), ('l', 'n'), ('m', 'o'), ('n', 'p'), ('o', 'q'), ('p', 'r'), (
'q', 's'), ('r', 't'), ('s', 'u'), ('t', 'v'), ('u', 'w'), ('v',
'x'), ('w', 'y'), ('x', 'z'), ('y', 'a'), ('z', 'b'), (',', ',')
, ('.', '.'), (' ', ' '), ("'", "'"), ('(', '('), (')', ')')]
```

actually we can create a dict

```
>>> dict(zip(a, b))
{'t': 'v', 'g': 'i', 'b': 'd', 'i': 'k', ',': ',', 'v': 'x', 'u'
: 'w', 'd': 'f', 'e': 'g', 'h': 'j', 'm': 'o', "'": "'", '(': '('
, '.': '.', 'q': 's', 'l': 'n', 'a': 'c', 'x': 'z', ' ': ' ', 'f'
: 'h', 'o': 'q', 'w': 'y', 'n': 'p', 'c': 'e', 'p': 'r', 's': 'u'
, 'z': 'b', 'j': 'l', 'y': 'a', 'r': 't', 'k': 'm', ')': ')'}
```

then mapping is as easy as

```
>>> dict(zip(a, b))['z']
'b'
```

translate the whole string:

```
>>> "".join([dict(zip(a,b))[x] for x in raw])
"i hope you didnt translate it by hand. thats what computers are
 for. doing itin by hand is inefficient and that's why this text
 is so long. using string.maketrans() is recommended. now apply
on the url."
```

Put Everything Together

```
raw = "g fmnc wms bgblr rpylqjyrc gr zw fylb. rfyrq ufyr amknsrc
pq ypc dmp. bmgle grgl zw fylb gq glcddgagclr ylb rfyr'q ufw rfg
q rcvr gq qm jmle. sqgle qrpgle.kyicrpylq() gq pcamkkclbcb. lmu
ynnjw ml rfc spj."

a = "abcdefghijklmnopqrstuvwxyz,. '()"
b = "cdefghijklmnopqrstuvwxyzab,. '()"

print("".join([dict(zip(a,b))[x] for x in raw]))
```

Next Level

As hinted, apply the same function on the url(`map`), we get `ocr` .

```
>>> "map".translate(str.maketrans("abcdefghijklmnopqrstuvwxyz",
"cdefghijklmnopqrstuvwxyzab"))
'ocr'
```

http://www.pythonchallenge.com/pc/def/ocr.html

Python 2 to Python 3

In Python 2, `maketrans()` is a method in `string` module, so you need to import it first:

```
import string

table = string.maketrans("from", "to")
```

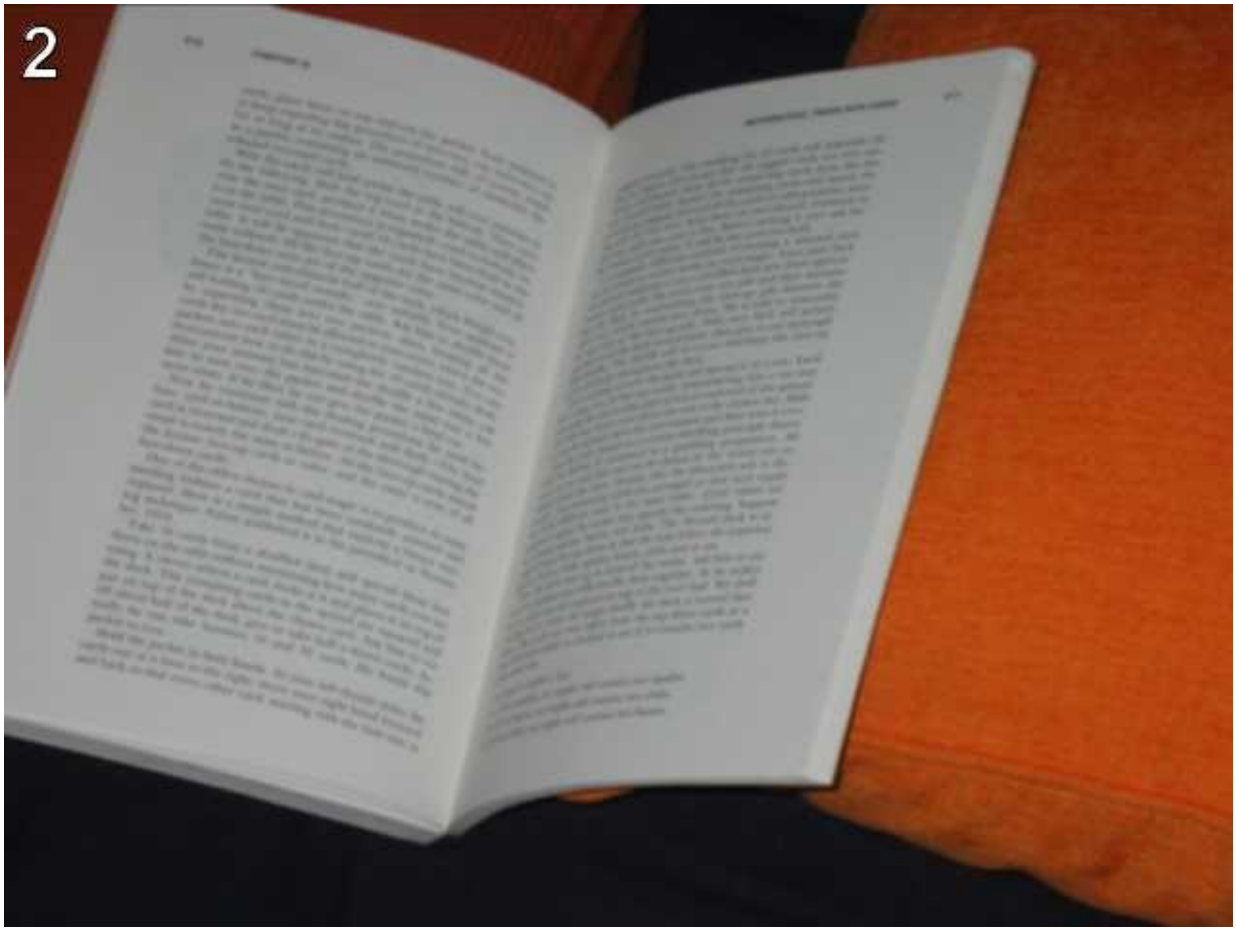
In Python 3, `maketrans()` is part of either `bytes` or `str` :

- `str.maketrans(from, to)`
- `bytes.maketrans(from, to)`

Python Challenge - Level 2

- Link: http://www.pythonchallenge.com/pc/def/ocr.html

Problem



> recognize the characters. maybe they are in the book, but MAYBE they are in the page source.

Right click and select "View Page Source":

```
<!--
find rare characters in the mess below:
-->

<!--
%%$@_$^__#)^)&!_+]!*@&^}@[@%]()%+$&[(_@%+%$*^@$^!+]!&_#)_*}{}}!}
_]$[%}@[{_@#_^{*
@##&{#&{&)*%(]{{([*}@[@&]+!!*{)!}{%+{))])[!^})+)$]#{*+^((@^@}$[*
*$&^{$!@#$%)!@(&
+^!{%_$&@^!}$_${)$_#)!({@!)(^}!*^&!$%_&&}&_#&@{)]{+)%*{&*%*&@%$+
]!*__(#!*){%&@++
!_)^$&&%#+)}!@!)&^}**#!_$([$!$}#*^}$+&#[{*{}{((#$]{[$[$$()_#}!@}
^@_&%^*!){*^^_$^
]@}#%[%!^[^_})+@&}{@*!(@$%$^)}[_!}(*}#}#___}!](@_{{(*#%!%%+*)^+#
%}$+_]#}%!**#!^_
...
```

Solution

Load Data

You can manually copy-and-paste the text to a file(`resources/level2.txt` in source code), then read from it:

```
>>> data = open('resources/level2.txt').read()
```

> **Read More**
>
> More about Python 3 - IO

Or extract the text from HTML directly. First load raw html source coding using `urllib.request` :

```
>>> import urllib.request
>>> html = urllib.request.urlopen("http://www.pythonchallenge.co
m/pc/def/ocr.html").read().decode()
```

Then extract the comment blocks in html. Note that by default dot does not match `\n`, so we need to use `re .DOTALL` flag.

```
>>> import re
>>> comments = re.findall("<!--(.*?)-->", html, re.DOTALL)
```

Alternatively we can use this:

```
>>> comments = re.findall("<!--([\w\n]*?)-->", html)
```

The pattern `<!--(.*)-->` will capture all blocks inside `<!--` and `-->`. We only care about the last part, so

```
>>> data = comments[-1]
```

Find the Rare Characters

```
>>> count = {}
>>> for c in data:
...     count[c] = count.get(c, 0) + 1
...
>>> count
{'*': 6034, '$': 6046, 'i': 1, '!': 6079, '[': 6108, 'u': 1, 'e'
: 1, '@': 6157, '#': 6115, 't': 1, '(': 6154, '+': 6066, '&': 60
43, 'q': 1, 'l': 1, '%': 6104, '{': 6046, '}': 6105, 'a': 1, '^'
: 6030, ']': 6152, '\n': 1221, 'y': 1, '_': 6112, ')': 6186}
```

The `rare` characters only have 1 occurrence: t, i, u, e, l, y, q, a.

In the right order

If you are having a hard time to guess the meaning:

```
>>> import re
>>> "".join(re.findall("[A-Za-z]", data))
'equality'
```

Put Everything Together

```
import urllib.request
import re
html = urllib.request.urlopen("http://www.pythonchallenge.com/pc
/def/ocr.html").read().decode()
data = re.findall("<!--(.*?)-->", html, re.DOTALL)[-1]
print("".join(re.findall("[A-Za-z]", data)))
```

result:

> equality

Next Level

http://www.pythonchallenge.com/pc/def/equality.html

Python Challenge - Level 3

- Link: http://www.pythonchallenge.com/pc/def/equality.html

Problem

> One small letter, surrounded by EXACTLY three big bodyguards on each of its sides.

The image shows 3 big candles on each side of a small one. The description hints us to find **letters**! So open the page source, we see a comment block at the bottom of the page:

```
<!--
kAewtloYgcFQaJNhHVGxXDiQmzjfcpYbzxlWrVcqsmUbCunkfxZWDZjUZMiGq
...
```

Hmm, shall we find all segments of the pattern `AAAbCCC`?

Solution

Step 1: Load Data

Similar to level 2, You can manually copy-and-paste the text to a file(`resources/level3.txt` in source code), then read from it:

```
>>> data = open('resources/level3.txt').read()
```

Or extract the text from HTML directly.

```
>>> import urllib.request
>>> import re
>>> html = urllib.request.urlopen("http://www.pythonchallenge.com/pc/def/equality.html").read().decode()
>>> data = re.findall("<!--(.*?)-->", html, re.DOTALL)[-1]
```

Step 2: Find the Matches

Now we have the content as a big long string, we can use regular expression to find all the matches. The pattern can be described as `[^A-Z]+[A-Z]{3}([a-z])[A-Z]{3}[^A-Z]+`. Here's a break down of the pattern:

- `[a-z]`: 1 lower case letter
- `[A-Z]`: 1 upper case letter
- `[A-Z]{3}`: 3 consecutive upper case letters
- `[A-Z]{3}[a-z][A-Z]{3}`: 3 upper case letters + 1 lower case letter + 3 upper case letters
- `[^A-Z]`: any character BUT an upper case letter
- `[^A-Z]+`: at least one such character
- `[^A-Z]+[A-Z]{3}[a-z][A-Z]{3}[^A-Z]+`: something else before and after our patter(`AAAbCCC`) so there's no more than 3 consecutive upper case letters on each side

- `[^A-Z]+[A-Z]{3}([a-z])[A-Z]{3}[^A-Z]+` : ...and we only care about the lower case

Let's see what we get:

```
>>> re.findall("[^A-Z]+[A-Z]{3}([a-z])[A-Z]{3}[^A-Z]+", data)
['l', 'i', 'n', 'k', 'e', 'd', 'l', 'i', 's', 't']
```

And join them together

```
>>> "".join(re.findall("[^A-Z]+[A-Z]{3}([a-z])[A-Z]{3}[^A-Z]+",
data))
'linkedlist'
```

That's it! **linkedlist**.

Put Everything Together

```
import urllib.request
import re

html = urllib.request.urlopen("http://www.pythonchallenge.com/pc
/def/equality.html").read().decode()
data = re.findall("<!--(.*)-->", html, re.DOTALL)[-1]
print("".join(re.findall("[^A-Z]+[A-Z]{3}([a-z])[A-Z]{3}[^A-Z]+"
, data)))
```

Next Level

http://www.pythonchallenge.com/pc/def/linkedlist.html

The page will redirect you to `linkedlist.php`

http://www.pythonchallenge.com/pc/def/linkedlist.php

Further Readings

- Python 3 - RegEx

Python Challenge - Level 4

- Link: http://www.pythonchallenge.com/pc/def/linkedlist.php

Problem

Click on the image, you would see

> and the next nothing is 44827

And the url changed to
`http://www.pythonchallenge.com/pc/def/linkedlist.php?nothing=12345`

Change the url with the new number and another number will be given.

Solution

First let's try to mimic the manual process we just tried:

```
>>> uri = "http://www.pythonchallenge.com/pc/def/linkedlist.php?
nothing=%s"
>>> num = "12345"
>>> content = urlopen(uri % num).read().decode()
>>> content
'and the next nothing is 44827'
```

Then use regular expression to extract the number:

```
>>> match = re.search("and the next nothing is (\d+)", content)
>>> match
<_sre.SRE_Match object; span=(0, 29), match='and the next nothin
g is 44827'>
```

Note `.group(0)` will return the whole text that matches the pattern, while the captured segments start from `.group (1)`

```
>>> match.group(0)
'and the next nothing is 44827'
>>> match.group(1)
'72198'
```

To automate this process, let's use a while loop, which stops when there's no match:

```
>>> pattern = re.compile("and the next nothing is (\d+)")
>>> while True:
...     content = urlopen(uri % num).read().decode('utf-8')
...     print(content)
...     match = pattern.search(content)
...     if match == None:
...         break
...     num = match.group(1)
...
and the next nothing is 44827
and the next nothing is 45439
<font color=red>Your hands are getting tired </font>and the next
 nothing is 94485
and the next nothing is 72198
...
and the next nothing is 16044
Yes. Divide by two and keep going.
```

Then reset `num` to `16044/2`

```
num = str(16044/2)
```

And re-run:

```
and the next nothing is 81226
and the next nothing is 4256
and the next nothing is 62734
and the next nothing is 25666
...
There maybe misleading numbers in the
text. One example is 82683. Look only for the next nothing and t
he next nothing is 63579
...
peak.html
```

Note that if you use `pattern.match()` instead of `pattern.search()` it would fail, since `.match()` is looking for a match starting from the very beginning, while there's one "misleading" line in the text.

Finally we have the answer: **peak.html**

Put Everything Together

```
from urllib.request import urlopen
import re

uri = "http://www.pythonchallenge.com/pc/def/linkedlist.php?noth
ing=%s"

num = "12345"
#num = str(16044/2)

pattern = re.compile("and the next nothing is (\d+)")

while True:
    content = urlopen(uri % num).read().decode('utf-8')
    print(content)
    match = pattern.search(content)
    if match == None:
        break
    num = match.group(1)
```

Result

```
peak.html
```

Next Level

http://www.pythonchallenge.com/pc/def/peak.html

Python 2 to Python 3

Python 2's `urllib` and `urllib2` are combined as the new `urllib`

Python Challenge - Level 5

- Link: http://www.pythonchallenge.com/pc/def/peak.html

Problem

> pronounce it

Page Source:

```
<peakhell src="banner.p"/>
<!-- peak hell sounds familiar ? -->
```

peakhell, *peakhell*... sounds like... "pickle"(which is Python's object serialization module)

Solution

Let's print out the raw data:

```
>>> from urllib.request import urlopen
>>> raw = urlopen("http://www.pythonchallenge.com/pc/def/banner.
p").read()
>>> raw
b"(lp0\n(lp1\n(S' '\np2\nI95\ntp3\naa..."
```

Normally plain text can be encoded as utf-8 or some other form, once we call `.decode()` it will reveal it self, however this one is tough:

```
>>> raw.decode()
"(lp0\n(lp1\n(S' '\np2\nI95\ntp3\naa..."
```

As hinted, let's use pickle to deserialize it:

```
>>> import pickle
>>> data = pickle.load(urlopen("http://www.pythonchallenge.com/p
c/def/banner.p"))
>>> data
[[(' ', 95)], [(' ', 14), ('#', 5), (' ', 70), ('#', 5), (' ', 1
)], [(' ', 15), ('#', 4), (' ', 71), ('#', 4), (' ', 1)], [(' ',
15), ('#', 4), (' ', 71), ('#', 4), (' ', 1)], [(' ', 15), ('#',
4), (' ', 71), ('#', 4), (' ', 1)], [(' ', 15), ('#', 4), (' ',
71), ('#', 4), (' ', 1)], [(' ', 15), ('#', 4), (' ', 71), ('#',
4), (' ', 1)], [(' ', 15), ('#', 4), (' ', 71), ('#', 4), (' ', 1
)], [(' ', 15), ('#', 4), (' ', 71), ('#', 4), (' ', 1)], [(' ',
6), ('#', 3), (' ', 6), ('#', 4), (' ', 3), ('#', 3), (' ', 9), (
'#', 3), (' ', 7), ('#', 5), (' ', 3), ('#', 3), (' ', 4), ('#',
5), (' ', 3), ('#', 3), (' ', 10), ('#', 3), (' ', 7), ('#', 4),
 (' ', 1)], [(' ', 3), ('#', 3), (' ', 3), ('#', 2), (' ', 4), (
'#', 4), (' ', 1), ('#', 7), (' ', 5), ('#', 2), (' ', 2), ('#',
3), (' ', 6), ('#', 4), (' ', 1), ('#', 7), (' ', 3), ('#', 4), (
' ', 1), ('#', 7), (' ', 5), ('#', 3), (' ', 2), ('#', 3), (' ',
5), ('#', 4), (' ', 1)], [(' ', 2), ('#', 3), (' ', 5), ('#', 3)
, (' ', 2), ('#', 5), (' ', 4), ('#', 4), (' ', 3), ('#', 3), ('
```

```
 ', 3), ('#', 4), (' ', 4), ('#', 5), (' ', 4), ('#', 4), (' ', 2
), ('#', 5), (' ', 4), ('#', 4), (' ', 3), ('#', 3), (' ', 5), (
'#', 3), (' ', 3), ('#', 4), (' ', 1)], [(' ', 1), ('#', 3), (' '
, 11), ('#', 4), (' ', 5), ('#', 4), (' ', 3), ('#', 3), (' ', 4
), ('#', 3), (' ', 4), ('#', 4), (' ', 5), ('#', 4), (' ', 2), (
'#', 4), (' ', 5), ('#', 4), (' ', 2), ('#', 3), (' ', 6), ('#',
4), (' ', 2), ('#', 4), (' ', 1)], [(' ', 1), ('#', 3), (' ', 11
), ('#', 4), (' ', 5), ('#', 4), (' ', 10), ('#', 3), (' ', 4), (
'#', 4), (' ', 5), ('#', 4), (' ', 2), ('#', 4), (' ', 5), ('#',
4), (' ', 2), ('#', 3), (' ', 7), ('#', 3), (' ', 2), ('#', 4), (
' ', 1)], [('#', 4), (' ', 11), ('#', 4), (' ', 5), ('#', 4), ('
 ', 5), ('#', 2), (' ', 3), ('#', 3), (' ', 4), ('#', 4), (' ', 5
), ('#', 4), (' ', 2), ('#', 4), (' ', 5), ('#', 4), (' ', 1), (
'#', 4), (' ', 7), ('#', 3), (' ', 2), ('#', 4), (' ', 1)], [('#'
, 4), (' ', 11), ('#', 4), (' ', 5), ('#', 4), (' ', 3), ('#', 10
), (' ', 4), ('#', 4), (' ', 5), ('#', 4), (' ', 2), ('#', 4), (
' ', 5), ('#', 4), (' ', 1), ('#', 14), (' ', 2), ('#', 4), (' '
, 1)], [('#', 4), (' ', 11), ('#', 4), (' ', 5), ('#', 4), (' ',
2), ('#', 3), (' ', 4), ('#', 4), (' ', 4), ('#', 4), (' ', 5), (
'#', 4), (' ', 2), ('#', 4), (' ', 5), ('#', 4), (' ', 1), ('#',
4), (' ', 12), ('#', 4), (' ', 1)], [('#', 4), (' ', 11), ('#', 4
), (' ', 5), ('#', 4), (' ', 1), ('#', 4), (' ', 5), ('#', 3), (
' ', 4), ('#', 4), (' ', 5), ('#', 4), (' ', 2), ('#', 4), (' ',
5), ('#', 4), (' ', 1), ('#', 4), (' ', 12), ('#', 4), (' ', 1)]
, [(' ', 1), ('#', 3), (' ', 11), ('#', 4), (' ', 5), ('#', 4), (
' ', 1), ('#', 4), (' ', 5), ('#', 3), (' ', 4), ('#', 4), (' ',
5), ('#', 4), (' ', 2), ('#', 4), (' ', 5), ('#', 4), (' ', 2), (
'#', 3), (' ', 12), ('#', 4), (' ', 1)], [(' ', 2), ('#', 3), ('
 ', 6), ('#', 2), (' ', 2), ('#', 4), (' ', 5), ('#', 4), (' ', 2
), ('#', 3), (' ', 4), ('#', 4), (' ', 4), ('#', 4), (' ', 5), (
'#', 4), (' ', 2), ('#', 4), (' ', 5), ('#', 4), (' ', 3), ('#',
3), (' ', 6), ('#', 2), (' ', 3), ('#', 4), (' ', 1)], [(' ', 3)
, ('#', 3), (' ', 4), ('#', 2), (' ', 3), ('#', 4), (' ', 5), ('
#', 4), (' ', 3), ('#', 11), (' ', 3), ('#', 4), (' ', 5), ('#',
4), (' ', 2), ('#', 4), (' ', 5), ('#', 4), (' ', 4), ('#', 3), (
' ', 4), ('#', 2), (' ', 4), ('#', 4), (' ', 1)], [(' ', 6), ('#'
, 3), (' ', 5), ('#', 6), (' ', 4), ('#', 5), (' ', 4), ('#', 2)
, (' ', 4), ('#', 4), (' ', 1), ('#', 6), (' ', 4), ('#', 11), (
' ', 4), ('#', 5), (' ', 6), ('#', 3), (' ', 6), ('#', 6)], [(' '
, 95)]]
```

OK! So data is a list of lists of tuples... It does not look like a **banner**(the file name), but more like the result of a simple string compression: convert the repetitive characters to tuples of (character, number of appearance). This level is all about serialization and compression, isn't it. Let's unravel the string:

```
>>> for line in data:
...     print("".join([k * v for k, v in line]))
...

              #####
                    #####
               ####
                     ####
               ####
                     ####
               ####
                     ####
               ####
                     ####
               ####
                     ####
               ####
                     ####
               ####
                     ####
      ###      ####   ###         ###       #####   ###    #####
   ###          ###       ####
   ###   ##    #### #######     ##  ###      #### #######   ####
 #######     ###  ###     ####
  ###     ###  #####    ####   ###   ####    #####    ####  ####
#    ####   ###     ###   ####
 ###           ####     ####   ###    ###    ####     ####  ####
     ####  ###      ####  ####
 ###           ####     ####          ###    ####     ####  ####
     ####  ###       ###  ####
####           ####     ####     ##   ###    ####     ####  ####
     #### ####       ###  ####
####           ####     ####   ##########    ####     ####  ####
```

```
      #### ##############  ####
####            ####     ####  ###    ####    ####     ####  ####
      #### ####            ####
####            ####     #### ####     ###    ####     ####  ####
      #### ####            ####
 ###            ####     #### ####     ###    ####     ####  ####
      ####  ###            ####
  ###       ##  ####     ####  ###    ####    ####     ####  ####
      ####   ###       ##   ####
   ###     ##   ####     ####   ############   ####     ####  ####
      ####    ###    ##    ####
       ###     ######    #####    ##    #### ######    ##########
#     #####       ###      ######
`
```

You got it:

> channel

Put Everything Together

```
from urllib.request import urlopen
import pickle

h = urlopen("http://www.pythonchallenge.com/pc/def/banner.p")
data = pickle.load(h)

for line in data:
    print("".join([k * v for k, v in line]))
```

Next Level

http://www.pythonchallenge.com/pc/def/channel.html

Python Challenge - Level 6

- Link: http://www.pythonchallenge.com/pc/def/channel.html

Problem

Pants? And Spiderman's underwear?

There's no text or any useful info in the source comments except for a call out of the donation button, but the big hint is at the very first line of html(and I missed it...):

```
<html> <!-- <-- zip -->
```

Ok... the image was about "zip", not "pants" or anything under it...

Replace `html` with `zip` : http://www.pythonchallenge.com/pc/def/channel.zip

Unzip it. In readme.txt:

```
welcome to my zipped list.

hint1: start from 90052
hint2: answer is inside the zip
```

Solution

```
>>> import zipfile, re
>>>
>>> f = zipfile.ZipFile("resources/channel.zip")
>>> num = '90052'
>>> while True:
...     content = f.read(num + ".txt").decode("utf-8")
...     print(content)
...     match = re.search("Next nothing is (\d+)", content)

...     if match == None:
...         break
...     num = match.group(1)
...
Next nothing is 94191
Next nothing is 85503
Next nothing is 70877
...
Next nothing is 68628
Next nothing is 67824
Next nothing is 46145
Collect the comments.
```

Comments.. what comments?

It turns out that zip file may contain some comments, and they can be accessed by:

- ZipFile.comment: comment associated with the ZIP file.
- ZipInfo.comment: comment for the individual archive member.

Add a few lines to collect the comments:

```
>>> num = '90052'
>>> comments = []
>>> while True:
...     content = f.read(num + ".txt").decode("utf-8")
...     comments.append(f.getinfo(num + ".txt").comment.decode("
utf-8"))
...     content
...     match = re.search("Next nothing is (\d+)", content)

...     if match == None:
...         break
...     num = match.group(1)
...
>>> print("".join(comments))
**************************************************************
**************************************************************
**                                                          **
**   OO    OO    XX      YYYY    GG    GG  EEEEEE NN      NN  **
**   OO    OO  XXXXXX   YYYYYY   GG   GG   EEEEEE  NN    NN   **
**   OO    OO XXX  XXX YYY   YY  GG GG     EE       NN  NN    **
**   OOOOOOOO XX    XX YY        GGG       EEEEE     NNNN     **
**   OOOOOOOO XX    XX YY        GGG       EEEEE      NN      **
**   OO    OO XXX  XXX YYY   YY  GG GG     EE         NN      **
**   OO    OO  XXXXXX   YYYYYY   GG   GG   EEEEEE     NN      **
**   OO    OO    XX      YYYY    GG    GG  EEEEEE     NN      **
**                                                          **
**************************************************************
 ************************************************************
```

If you try http://www.pythonchallenge.com/pc/def/hockey.html, you will get

```
it's in the air. look at the letters.
```

The right answer is in the letters: **oxygen**

Put Everything Together

```
import zipfile, re

f = zipfile.ZipFile("channel.zip")
print(f.read("readme.txt").decode("utf-8"))

num = '90052'

comments = []

while True:
    content = f.read(num + ".txt").decode("utf-8")
    comments.append(f.getinfo(num + ".txt").comment.decode("utf-
8"))
    print(content)
    match = re.search("Next nothing is (\d+)", content)
    if match == None:
        break
    num = match.group(1)

print("".join(comments))
```

Next Level

http://www.pythonchallenge.com/pc/def/oxygen.html

Python Challenge - Level 7

- Link: http://www.pythonchallenge.com/pc/def/oxygen.html

Problem

Nothing else... the gray scale should contain the information.

We need an image processing library.

Solution 1: PIL(Pillow)

> ### Python 2 to 3
>
> The original PIL was not ported to Python 3. Use the fork called ``Pillow`` instead

Install Pillow

```
$ pip3 install pillow
```

Make sure you can load the module:

```
>>> from PIL import Image
```

To load an image, one way is to download the image to local manually, then load it by:

```
>>> img = Image.open("oxygen.png")
```

Or load it from url directly:

```
>>> import requests
>>> from io import BytesIO
>>> from PIL import Image
>>> img = Image.open(BytesIO(requests.get('http://www.pythonchal
lenge.com/pc/def/oxygen.png').content))
```

We can get some basic info about the image:

```
>>> im.width
629
>>> im.height
95
```

And get the pixel by providing indices:

```
>>> img.getpixel((0,0))
(79, 92, 23, 255)
```

The result is the tuple of (R, G, B, alpha).

To get the grey scale, we can take the middle row of the pixels:

```
>>> row = [img.getpixel((x, img.height / 2)) for x in range(img.
width)]
>>> row
[(115, 115, 115, 255), (115, 115, 115, 255), (115, 115, 115, 255
), ...
```

As you can tell, row has lots of duplicates, since each grey block's width is larger than 1 pixel. If you do some manual counting, you know it is exactly 7 pixels wide, this should be the easiest way to de-dup:

```
>>> row = row[::7]
>>> row
[(115, 115, 115, 255), (109, 109, 109, 255), (97, 97, 97, 255),
...
```

Notice that at the end of the array there are some noises: pixels that are not grey scale, which have the same value for R, G, and B. We can remove those pixels

```
>>> ords = [r for r, g, b, a in row if r == g == b]
```

and since RGB is using a positive number in [0, 255] for each color, we can assume it represents a ASCII character:

```
>>> "".join(map(chr, ords))
'smart guy, you made it. the next level is [105, 110, 116, 101,
103, 114, 105, 116, 121]'
```

We were right, but it is not over... Do it again on the numbers:

```
>>> nums = re.findall("\d+", "".join(map(chr, ords)))
>>> nums
['105', '110', '116', '101', '103', '114', '105', '116', '121']
```

Finally:

```
>>> "".join(map(chr, map(int, nums)))
'integrity'
```

Solution 2: PyPNG

Alternatively use a package called `PyPNG` :

```
pip install pypng
```

```
from urllib.request import urlopen
import png, re

response = urlopen("http://www.pythonchallenge.com/pc/def/oxygen
.png")

(w, h, rgba, dummy) = png.Reader(response).read()

it = list(rgba)
mid = int(h / 2)
l = len(it[mid])
res = [chr(it[mid][i]) for i in range(0, l, 4*7) if it[mid][i] =
= it[mid][i + 1] == it[mid][i + 2]]
print("".join(res))
```

The pixels are stored as `[r, g, b, a, r, g, b, a...]`, if the pixel is gray, rgb should be equivalent. Another tricky part is the width of each block is 7px(correspond to the number of level?) so the step of the range is `4 * 7`.

Output

> smart guy, you made it. the next level is [105, 110, 116, 101, 103, 114, 105, 116, 121]

Modify the last line:

```
print("".join(map(chr, map(int, re.findall("\d+", "".join(res)))
)))
```

So integers are extracted and mapped to characters

Final result:

> integrity

Next Level

http://www.pythonchallenge.com/pc/def/integrity.html

Read More

- Pillow Documentation

Python Challenge - Level 8

- Link: http://www.pythonchallenge.com/pc/def/integrity.html

Problem

> Where is the missing link?

The fly is clickable, but it asks for user name and password, and this line:

> The site says: "inflate"

And in the page source:

```
<!--
un: 'BZh91AY&SYA\xaf\x82\r\x00\x00\x01\x01\x80\x02\xc0\x02\x00 \
x00!\x9ah3M\x07<]\xc9\x14\xe1BA\x06\xbe\x084'
pw: 'BZh91AY&SY\x94$|\x0e\x00\x00\x00\x81\x00\x03$ \x00!\x9ah3M\
x13<]\xc9\x14\xe1BBP\x91\xf08'
-->
```

The opposite of "inflate" is... "compress". Do not blame yourself if you do not know which compression format it is. Here are some clues that it is "bzip2"...

> .magic:16 = 'BZ' signature/magic number .version:8 = 'h' for Bzip2 ('H'uffman coding), '0' for Bzip1 (deprecated) .hundred_k_blocksize:8 = '1'..'9' block-size 100 kB-900 kB (uncompressed)

Solution

Let's decompress it:

```
>>> import bz2
>>> bz2.decompress('BZh91AY&SYA\xaf\x82\r\x00\x00\x01\x01\x80\x0
2\xc0\x02\x00 \x00!\x9ah3M\x07<]\xc9\x14\xe1BA\x06\xbe\x084')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.5/bz2.py", line 348, in decompress
    res = decomp.decompress(data)
TypeError: a bytes-like object is required, not 'str'
```

That does not work... `.decompress()` expects some `bytes`, not string

```
>>> bz2.decompress(b'BZh91AY&SYA\xaf\x82\r\x00\x00\x01\x01\x80\x
02\xc0\x02\x00 \x00!\x9ah3M\x07<]\xc9\x14\xe1BA\x06\xbe\x084')
b'huge'
>>> bz2.decompress(b'BZh91AY&SY\x94$|\x0e\x00\x00\x00\x81\x00\x0
3$ \x00!\x9ah3M\x13<]\xc9\x14\xe1BBP\x91\xf08')
b'file'
```

Put Everything Together

```
import bz2

un = b'BZh91AY&SYA\xaf\x82\r\x00\x00\x01\x01\x80\x02\xc0\x02\x00
 \x00!\x9ah3M\x07<]\xc9\x14\xe1BA\x06\xbe\x084'
pw = b'BZh91AY&SY\x94$|\x0e\x00\x00\x00\x81\x00\x03$ \x00!\x9ah3
M\x13<]\xc9\x14\xe1BBP\x91\xf08'

print(bz2.decompress(un))
print(bz2.decompress(pw))
```

Next Level

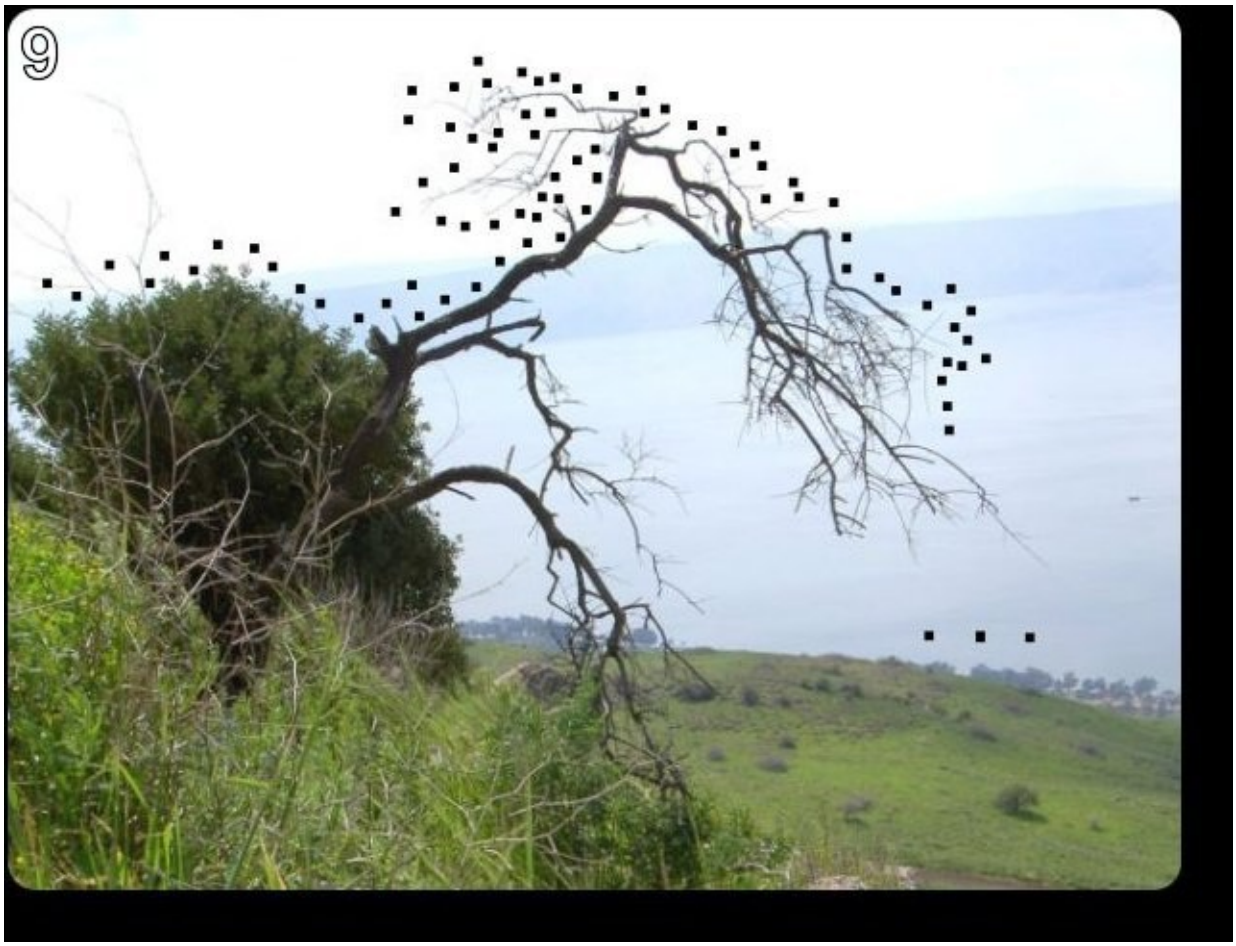
Click the fly(or the link in page source): `../return/good.html` , and fill in username(`huge`) and password(`file`)

http://www.pythonchallenge.com/pc/return/good.html

Python Challenge - Level 9

- Link: http://www.pythonchallenge.com/pc/return/good.html
- Username: **huge**
- Password: **file**

Problem



In page source:

```
<!--
first+second=?

first:
146,399,163,403,170,393,169,391,166,386,170,381,170,371,170,355,
169,346,167,335,170,329,170,320,170,
```

```
310,171,301,173,290,178,289,182,287,188,286,190,286,192,291,194,
296,195,305,194,307,191,312,190,316,
190,321,192,331,193,338,196,341,197,346,199,352,198,360,197,366,
197,373,196,380,197,383,196,387,192,
389,191,392,190,396,189,400,194,401,201,402,208,403,213,402,216,
401,219,397,219,393,216,390,215,385,
215,379,213,373,213,365,212,360,210,353,210,347,212,338,213,329,
214,319,215,311,215,306,216,296,218,
290,221,283,225,282,233,284,238,287,243,290,250,291,255,294,261,
293,265,291,271,291,273,289,278,287,
279,285,281,280,284,278,284,276,287,277,289,283,291,286,294,291,
296,295,299,300,301,304,304,320,305,
327,306,332,307,341,306,349,303,354,301,364,301,371,297,375,292,
384,291,386,302,393,324,391,333,387,
328,375,329,367,329,353,330,341,331,328,336,319,338,310,341,304,
341,285,341,278,343,269,344,262,346,
259,346,251,349,259,349,264,349,273,349,280,349,288,349,295,349,
298,354,293,356,286,354,279,352,268,
352,257,351,249,350,234,351,211,352,197,354,185,353,171,351,154,
348,147,342,137,339,132,330,122,327,
120,314,116,304,117,293,118,284,118,281,122,275,128,265,129,257,
131,244,133,239,134,228,136,221,137,
214,138,209,135,201,132,192,130,184,131,175,129,170,131,159,134,
157,134,160,130,170,125,176,114,176,
102,173,103,172,108,171,111,163,115,156,116,149,117,142,116,136,
115,129,115,124,115,120,115,115,117,
113,120,109,122,102,122,100,121,95,121,89,115,87,110,82,109,84,1
18,89,123,93,129,100,130,108,132,110,
133,110,136,107,138,105,140,95,138,86,141,79,149,77,155,81,162,9
0,165,97,167,99,171,109,171,107,161,
111,156,113,170,115,185,118,208,117,223,121,239,128,251,133,259,
136,266,139,276,143,290,148,310,151,
332,155,348,156,353,153,366,149,379,147,394,146,399

second:
156,141,165,135,169,131,176,130,187,134,191,140,191,146,186,150,
179,155,175,157,168,157,163,157,159,
157,158,164,159,175,159,181,157,191,154,197,153,205,153,210,152,
212,147,215,146,218,143,220,132,220,
125,217,119,209,116,196,115,185,114,172,114,167,112,161,109,165,
```

```
107,170,99,171,97,167,89,164,81,162,
77,155,81,148,87,140,96,138,105,141,110,136,111,126,113,129,118,
117,128,114,137,115,146,114,155,115,
158,121,157,128,156,134,157,136,156,136

-->
```

`first` and `second` have different lengths, so it is not simply adding up the two arrays. The image has some dots, maybe we should just connect the dots...

Solution

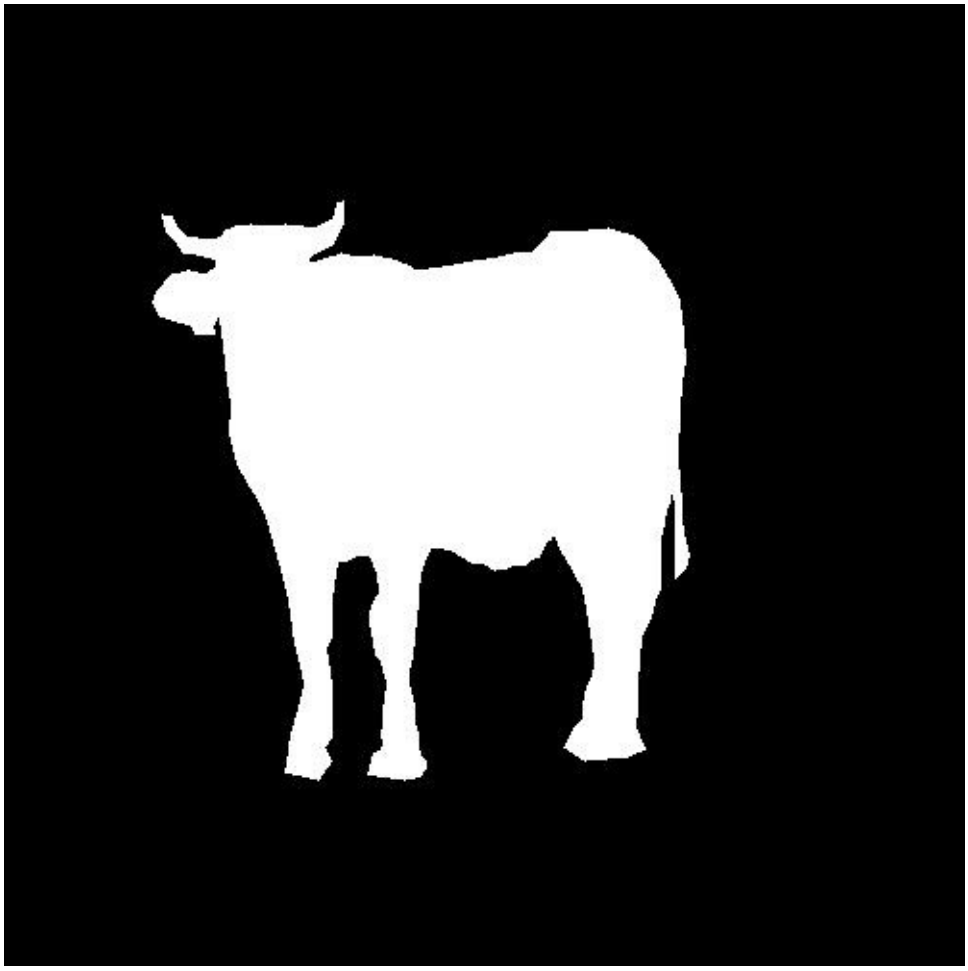
Copy and paste...

```
>>> first = [
... 146,399,163,403,170,393,169,391,166,386,170,381,170,371,170,
355,169,346,167,335,170,329,170,320,170,
... 310,171,301,173,290,178,289,182,287,188,286,190,286,192,291,
194,296,195,305,194,307,191,312,190,316,
... 190,321,192,331,193,338,196,341,197,346,199,352,198,360,197,
366,197,373,196,380,197,383,196,387,192,
... 389,191,392,190,396,189,400,194,401,201,402,208,403,213,402,
216,401,219,397,219,393,216,390,215,385,
... 215,379,213,373,213,365,212,360,210,353,210,347,212,338,213,
329,214,319,215,311,215,306,216,296,218,
... 290,221,283,225,282,233,284,238,287,243,290,250,291,255,294,
261,293,265,291,271,291,273,289,278,287,
... 279,285,281,280,284,278,284,276,287,277,289,283,291,286,294,
291,296,295,299,300,301,304,304,320,305,
... 327,306,332,307,341,306,349,303,354,301,364,301,371,297,375,
292,384,291,386,302,393,324,391,333,387,
... 328,375,329,367,329,353,330,341,331,328,336,319,338,310,341,
304,341,285,341,278,343,269,344,262,346,
... 259,346,251,349,259,349,264,349,273,349,280,349,288,349,295,
349,298,354,293,356,286,354,279,352,268,
... 352,257,351,249,350,234,351,211,352,197,354,185,353,171,351,
154,348,147,342,137,339,132,330,122,327,
... 120,314,116,304,117,293,118,284,118,281,122,275,128,265,129,
```

```
257,131,244,133,239,134,228,136,221,137,
... 214,138,209,135,201,132,192,130,184,131,175,129,170,131,159,
134,157,134,160,130,170,125,176,114,176,
... 102,173,103,172,108,171,111,163,115,156,116,149,117,142,116,
136,115,129,115,124,115,120,115,115,117,
... 113,120,109,122,102,122,100,121,95,121,89,115,87,110,82,109,
84,118,89,123,93,129,100,130,108,132,110,
... 133,110,136,107,138,105,140,95,138,86,141,79,149,77,155,81,1
62,90,165,97,167,99,171,109,171,107,161,
... 111,156,113,170,115,185,118,208,117,223,121,239,128,251,133,
259,136,266,139,276,143,290,148,310,151,
... 332,155,348,156,353,153,366,149,379,147,394,146,399]
>>> second = [156,141,165,135,169,131,176,130,187,134,191,140,191
,146,186,150,179,155,175,157,168,157,163,157,159,
... 157,158,164,159,175,159,181,157,191,154,197,153,205,153,210,
152,212,147,215,146,218,143,220,132,220,
... 125,217,119,209,116,196,115,185,114,172,114,167,112,161,109,
165,107,170,99,171,97,167,89,164,81,162,
... 77,155,81,148,87,140,96,138,105,141,110,136,111,126,113,129,
118,117,128,114,137,115,146,114,155,115,
... 158,121,157,128,156,134,157,136,156,136]
```

Use PIL(pillow)

```
>>> from PIL import Image,ImageDraw
>>> im = Image.new('RGB', (500,500))
>>> draw = ImageDraw.Draw(im)
>>> draw.polygon(first, fill='white')
>>> draw.polygon(second, fill='white')
>>> im.show()
```



Is it a ... cow? We get this from `cow.html`

> hmm. it's a male.

Bull? It works...

Next Level

http://www.pythonchallenge.com/pc/return/bull.html

Python Challenge - Level 10

- Link: http://www.pythonchallenge.com/pc/return/bull.html
- Username: **huge**
- Password: **file**

Problem

> len(a[30]) = ?

By clicking the image:

```
a = [1, 11, 21, 1211, 111221,
```

A look-and-say sequence

Solution 1

This is a naive solution, just look, and say, assemble the string while we are counting.

```
a = '1'
b = ''
for i in range(0, 30):
    j = 0
    k = 0
    while j < len(a):
        while k < len(a) and a[k] == a[j]: k += 1
        b += str(k-j) + a[j]
        j = k
    print b
    a = b
    b = ''
print len(a)
```

Solution 2

Python can do much better. We can use regular expression to find the (number, length) pairs

```
>>> import re
>>> re.findall(r"(\d)(\1*)", "111221")
[('1', '11'), ('2', '2'), ('1', '')]
```

Note that the pattern is a *raw string*(`r"..."`), which means backslash(`\`) does not need to be escaped. It is equivalent to

```
>>> re.findall("(\\d)(\\1*)", "111221")
```

The result tuples are in the form: (first appearance, following appearance), so from the first one we get the number, and the second one we get the length(remember to +1)

```
>>> "".join([str(len(i+j))+i for i,j in re.findall(r"(\d)(\1*)",
 x)])
'11'
>>> "".join([str(len(i+j))+i for i,j in re.findall(r"(\d)(\1*)",
"1")])
'11'
>>> "".join([str(len(i+j))+i for i,j in re.findall(r"(\d)(\1*)",
"11")])
'21'
>>> "".join([str(len(i+j))+i for i,j in re.findall(r"(\d)(\1*)",
"21")])
'1211'
>>> "".join([str(len(i+j))+i for i,j in re.findall(r"(\d)(\1*)",
"1211")])
'111221'
```

Let it run 30 times:

```
>>> x = "1"
>>> for n in range(30):
...     x="".join([str(len(j) + 1)+i for i, j in re.findall(r"(\
d)(\1*)", x)])
...
>>> len(x)
5808
```

Solution 3

Similar to Solution 2, but we are using `itertools.groupby()` instead of regular expression to find the (number, length) pairs:

```
>>> "".join([str(len(list(j))) + i for i,j in itertools.groupby(
"1211")])
'111221'
```

The result of `groupby` is (number, all appearances)

```
>>> [(i, list(j)) for i,j in itertools.groupby("1211")]
[('1', ['1']), ('2', ['2']), ('1', ['1', '1'])]
```

that is why we do not need to `+1` when calculating the length as in Solution 2.

```
>>> x = "1"
>>> for n in range(30):
...     x = "".join([str(len(list(j))) + i for i,j in itertools.
groupby(x)])
...
>>> len(x)
5808
```

Solution 4

This is not recommend, but it is a cool one-liner solution. Do not sacrifice clarity for coolness in the real world projects!

```
>>> from itertools import groupby
>>> from functools import reduce
>>> len(reduce(lambda x, n:reduce(lambda a, b: a + str(len(list(
b[1]))) + b[0], groupby(x), ""), range(30), "1"))
5808
```

2 nested `reduce()`, the outer one simply let it run for 30 times, and set the initial value `1` for `x`; the inner one does exactly the same as in Solution 3.

Again, whether this bothers you or not, do not code anything that need you extensive explanations.

Next Level

http://www.pythonchallenge.com/pc/return/5808.html

Python Challenge - Level 11

- Link: http://www.pythonchallenge.com/pc/return/5808.html
- Username: **huge**
- Password: **file**

Problem

title: `odd even`

Solution

The only clue is the title, which implies that we need to split the image by odd/even:

```
from PIL import Image

im = Image.open('cave.jpg')
(w, h) = im.size

even = Image.new('RGB', (w // 2, h // 2))
odd = Image.new('RGB', (w // 2, h // 2))

for i in range(w):
    for j in range(h):
        p = im.getpixel((i,j))
        if (i+j)%2 == 1:
            odd.putpixel((i // 2, j // 2), p)
        else:
            even.putpixel((i // 2, j // 2), p)
even.save('even.jpg')
odd.save('odd.jpg')
```

Open `even.jpg` you will see a word: evil

Next Level

http://www.pythonchallenge.com/pc/return/evil.html

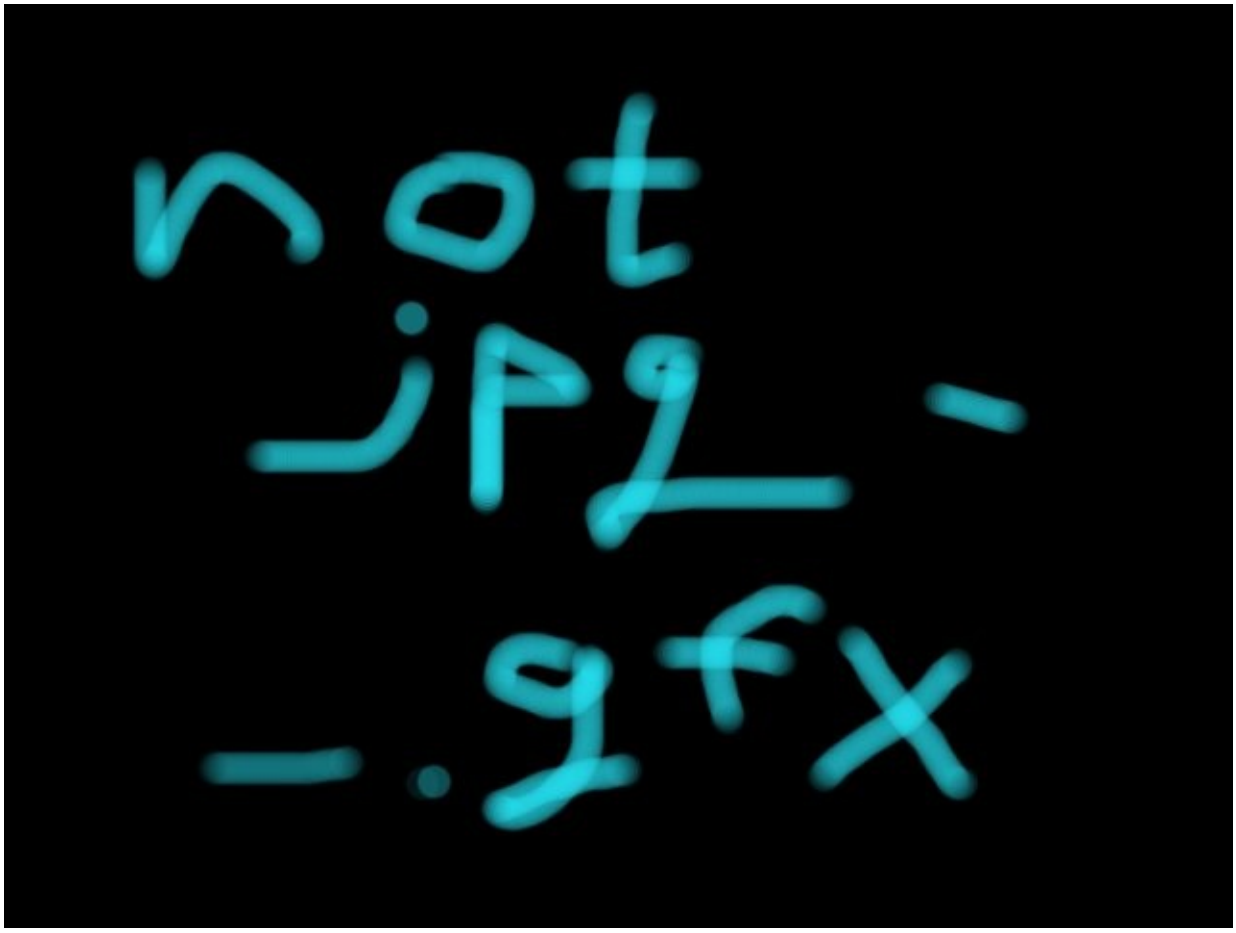
Python Challenge - Level 12

- Link: http://www.pythonchallenge.com/pc/return/evil.html
- Username: **huge**
- Password: **file**

Problem

Checking the source, the image is named as `evil1.jpg`. Try `evil2` and you will see:

http://www.pythonchallenge.com/pc/return/evil2.jpg



Change suffix to `.gfx` to download the file

Is it over? Try evil3.jpg, we got:

Really? "no more evils"? Let's try evil4.jpg. Depend on your browser, you may see nothing or an error message

> The image "http://www.pythonchallenge.com/pc/return/evil4.jpg" cannot be displayed because it contains errors.

But, it is not 404! Let's take a closer look:

```
$ curl -u huge:file http://www.pythonchallenge.com/pc/return/evi
l4.jpg
Bert is evil! go back!
```

OK, Bert is evil, whoever that is.

Solution

It is hard to tell what's going on inside the file.

```
>>> data = open("evil2.gfx", "rb").read()
>>> data
b'\xff\x89G\x89\xff\xd8PIP\xd8\xffNFN
```

Back to the problem's image, someone is dealing the cards into 5 stacks, and all the cards are ... 5.

```
>>> len(data)
67575
```

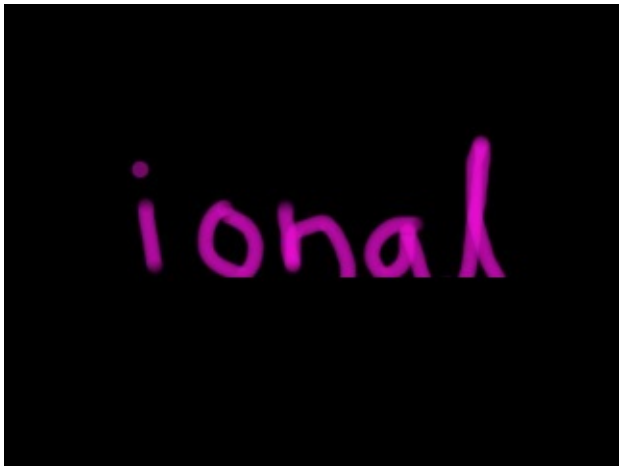
Let's try to split the data into 5, but "dealing" the bytes instead of cards...

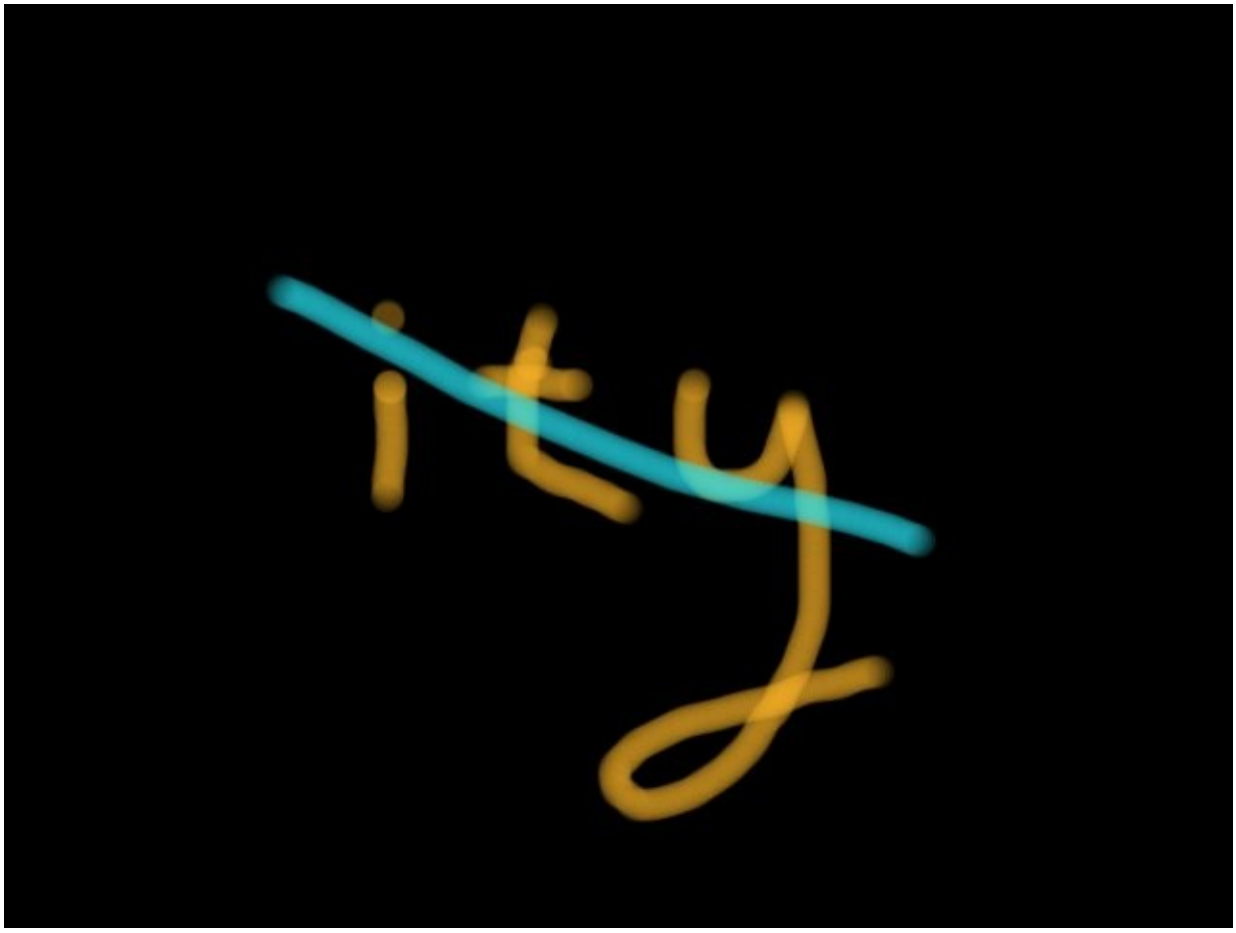
```
>>> for i in range(5):
...     open('%d.jpg' % i ,'wb').write(data[i::5])
```

Open the generated `0.jpg` through `4.jpg`, you should see `dis`, `pro`, `port`, `tional`


pro


port


ional



Put Everything Together

```
data = open("evil2.gfx", "rb").read()
for i in range(5):
    open('%d.jpg' % i ,'wb').write(data[i::5])
```

Next Level

http://www.pythonchallenge.com/pc/return/disproportional.html

Python Challenge - Level 13

- Link: http://www.pythonchallenge.com/pc/return/disproportional.html
- Username: **huge**
- Password: **file**

Problem

> phone that evil

Key 5 is clickable,

```
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value>
            <int>105</int>
          </value>
        </member>
        <member>
          <name>faultString</name>
          <value>
            <string>
              XML error: Invalid document end at line 1, column
1
            </string>
          </value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Solution

We can use `xmlrpc` module to talk to it:

```
>>> import xmlrpc.client
>>> conn = xmlrpc.client.ServerProxy("http://www.pythonchallenge
.com/pc/phonebook.php")
>>> conn
<ServerProxy for www.pythonchallenge.com/pc/phonebook.php>
>>> conn.system.listMethods()
['phone', 'system.listMethods', 'system.methodHelp', 'system.met
hodSignature', 'system.multicall', 'system.getCapabilities']
```

`phone` seems to be the method we should call. Get more info on how to use it:

```
>>> conn.system.methodHelp("phone")
'Returns the phone of a person'
>>> conn.system.methodSignature("phone")
[['string', 'string']]
```

So it takes a string and returns a string, the input should be the name and the output should be the number:

```
>>> conn.phone("Bert")
'555-ITALY'
```

If you do not know, `555` basically means "fake phone numbers" in US... The key is "**italy**".

Put Everything Together

```
import xmlrpc.client

conn = xmlrpc.client.ServerProxy("http://www.pythonchallenge.com
/pc/phonebook.php")
print(conn.phone("Bert"))
```

Next Level

http://www.pythonchallenge.com/pc/return/italy.html

Python Challenge - Level 14

Problem Link

- http://www.pythonchallenge.com/pc/return/italy.html
- Username: **huge**
- Password: **file**

Clues

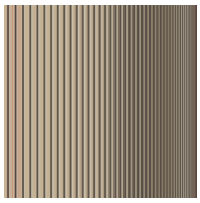
Clue 1

In HTML source code:

```
<!-- remember: 100*100 = (100+99+99+98) + (...  -->

<img src="wire.png" width="100" height="100">
```

This is the image:



Clue 2

Title of the HTML:

> walk around

Clue 3

The image of the problem: we are looking at(for) something spiral!

Exploration

Do not be fooled by that barcode-like image, it appears to be a square, that is because in HTML both the width and height are set as 100.

```
>>> from PIL import Image
>>> im = Image.open('../images/wire.png')
>>> im.size
(10000, 1)
```

The real size is not `100*100` ! It is `10000*1` !

Based on all the clues, our assumption is to use the 10000 points from the given image, create another image of 100*100, by walking around the square from outside to inside, so we go right for 100 pixels, then down 99 pixels, then left 99 pixels, then up 98 pixels(clue 1).

Solution

```
from PIL import Image
im = Image.open('wire.png')
delta = [(1,0),(0,1),(-1,0),(0,-1)]
out = Image.new('RGB', [100,100])
x,y,p = -1,0,0
d = 200
while d/2>0:
    for v in delta:
        steps = d // 2
        for s in range(steps):
            x, y = x + v[0], y + v[1]
            out.putpixel((x, y),im.getpixel((p,0)))
            p += 1
        d -= 1
out.save('level_14_result.jpg')
```

It's a cat!

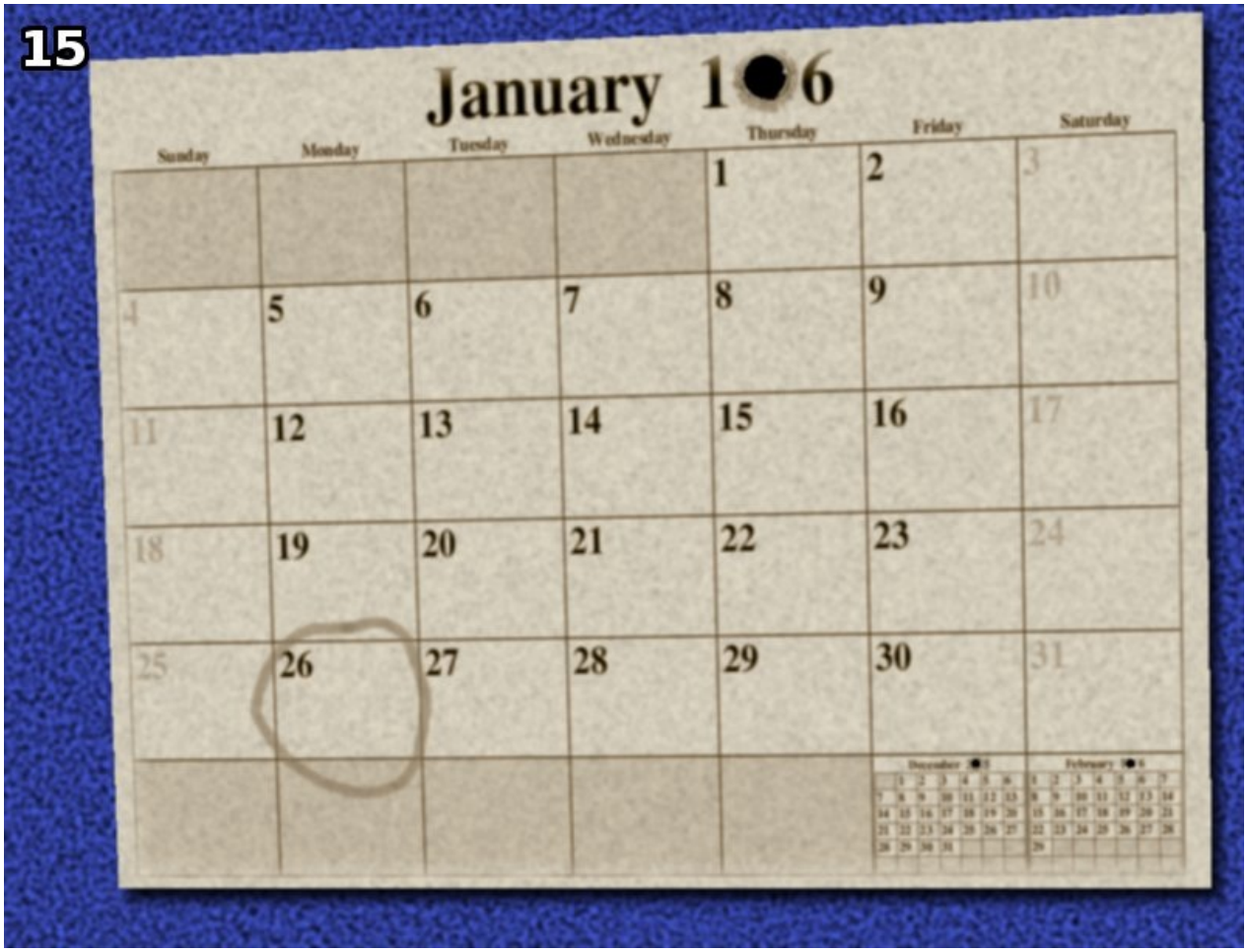
Next Level

http://www.pythonchallenge.com/pc/return/cat.html

and its name is uzi. you'll hear from him later.

http://www.pythonchallenge.com/pc/return/uzi.html

Python Challenge - Level 15



Problem Link

- Link: http://www.pythonchallenge.com/pc/return/uzi.html
- Username: **huge**
- Password: **file**

Clues

Clue 1

There's a burnt hole on the calendar, it must be year 1xx6.

Clue 2

That year's Jan 1 is a Thursday.

Clue 3

It was a leap year, since Feb 29 is on the calendar.

Clue 4

We are looking for Jan 26.

Clue 5

In HTML source code:

```
<!-- he ain't the youngest, he is the second -->
```

Clue 6

Title: `whom?`

Clue 7

In HTML source code:

```
<!-- todo: buy flowers for tomorrow -->
```

Exploration

So we are looking for a person, who's related to that particular date, and he is not the youngest...

The key is to find what year it was. We can filter the years between 1006 and 1996 by those clues.

First the leap year. Is 1006 a leap year?

```
>>> 1006 / 4
251.5
```

No, it is not, so our starting point should be 1016:

```
>>> 1016 / 4
254.0
```

Then we can test 1036, 1056, 1076... every 20 years.

```
>>> list(range(1016, 1996, 20))
[1016, 1036, 1056, 1076, 1096, 1116, 1136, 1156, 1176, 1196, 1216
, 1236, 1256, 1276, 1296, 1316, 1336, 1356, 1376, 1396, 1416, 14
36, 1456, 1476, 1496, 1516, 1536, 1556, 1576, 1596, 1616, 1636,
1656, 1676, 1696, 1716, 1736, 1756, 1776, 1796, 1816, 1836, 1856
, 1876, 1896, 1916, 1936, 1956, 1976]
```

Then find the year that has Jan 26 as a Monday

```
>>> import datetime
>>> [year for year in range(1016, 1996, 20) if datetime.date(yea
r, 1, 26).isoweekday() == 1]
[1176, 1356, 1576, 1756, 1976]
```

From clue 5:

> he ain't the youngest, he is the second

1976 must be the youngest, so 1756 is the second youngest... what is special with 1756-01-26? Nothing. But remember the last clue:

> todo: buy flowers for tomorrow

And... 1756-01-27 is the birthday of Mozart...

Solution

Alternatively we can use `canlendar.isleap()` to detect leap year, which makes it more clear. Check out the full solution:

```
import datetime, calendar
for year in range(1006,1996,10):
    d = datetime.date(year, 1, 26)
    if d.isoweekday() == 1 & calendar.isleap(year):
        print(d)
```

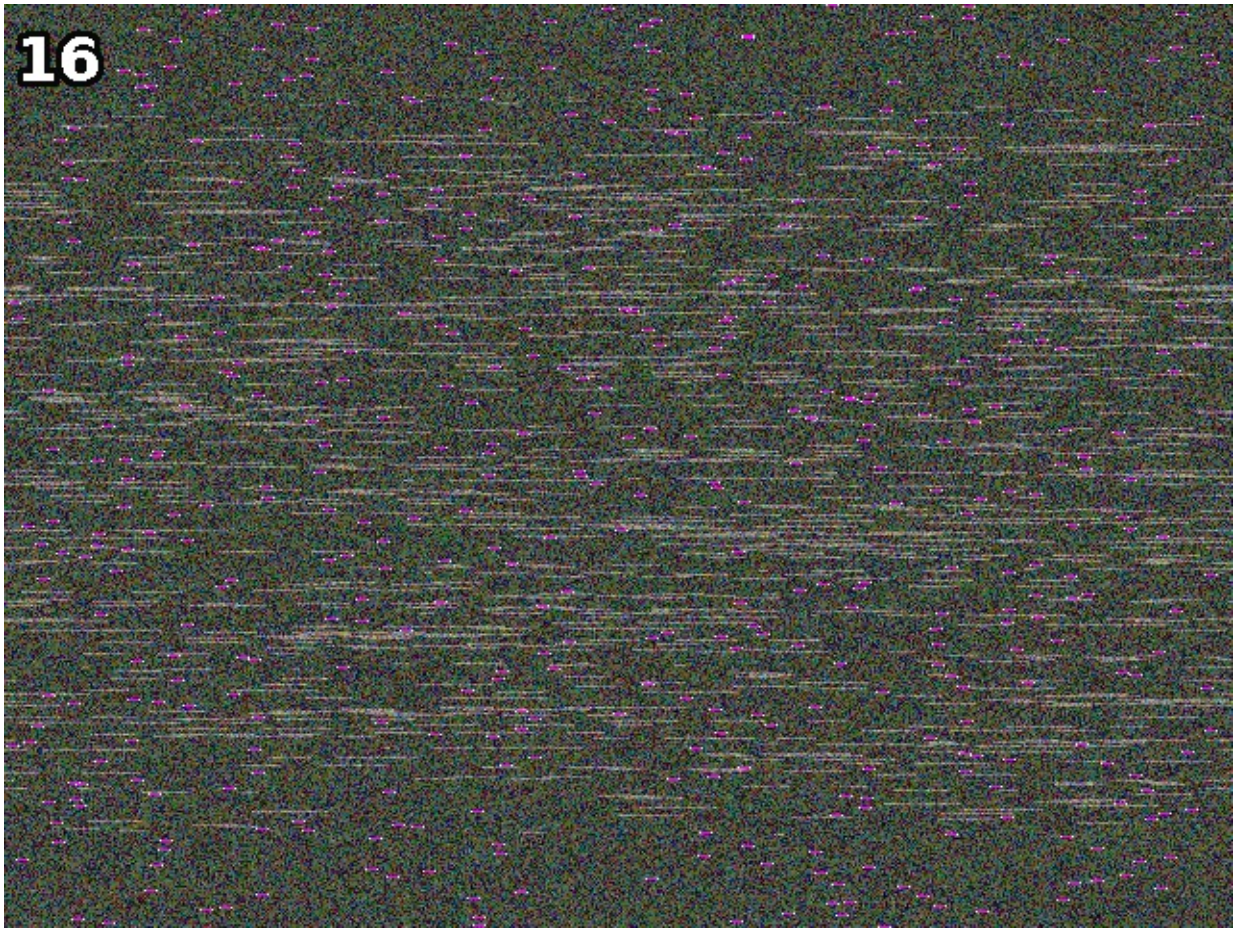
Output

```
1176-01-26
1356-01-26
1576-01-26
1756-01-26
1976-01-26
```

Next Level

http://www.pythonchallenge.com/pc/return/mozart.html

Python Challenge - Level 16



Problem Link

- Link: http://www.pythonchallenge.com/pc/return/mozart.html
- Username: **huge**
- Password: **file**

Clues

Clue 1

Page Title:

> let me get this straight

Clue 2

Image: There are pink segments, looks like they have the same length.

Exploration

Sounds like we need to align the pink segments. But how do you define "pink"

Find the Pink Segments

First load the image:

```
from PIL import Image, ImageChops

image = Image.open("mozart.gif")
```

Could it be the most frequent pixels? `image.histogram()` can give us the counts of each pixel.

```
>>> max(enumerate(image.histogram()), key=lambda x: x[1])
(60, 29448)
```

So value 60 appears 29448 times, let's see if it is pink. We can make a copy of the current image(to use the same palette), and paint all the pixels as 60:

```
>>> tmp = image.copy()
>>> tmp.frombytes(bytes([60] * (tmp.height * tmp.width)))
>>> tmp.show()
```

Hmm, not so pinky.

If pink segments exists in every row, then the number of pink pixels should be divisible by height of the image, right?

```
>>> [x for x in image.histogram() if x % image.height == 0 and x
 != 0]
[2400]
```

Good, there's only one!

```
>>> image.histogram().index(2400)
195
```

Let's try pixel value of 195 instead of 60

```
>>> tmp.frombytes(bytes([195] * (tmp.height * tmp.width)))
>>> tmp.show()
```

Now it looks right.

Shift The Rows

Let's shift each row:

```
import numpy as np
>>> shifted = [bytes(np.roll(row, -row.tolist().index(195)).toli
st()) for row in np.array(image)]
```

We are using `np.array(image)` to load image data, alternatively you can use `list(image.getdata())`, however you will get a one-dimension array, so you need to reshape the array manually.

We take each `row` in the ndarray, and get the first position of the pink pixel by `row.tolist().index(195)`, then `np.roll()` can help us shift the array.

Once we have the shifted data as `bytes`, we can use `Image.frombytes()` to create a new image, while reusing the original image's mode and size.

```
>>> Image.frombytes(image.mode, image.size, b"".join(shifted)).s
how()
```

Here's the result:

Solution

Solution 1

```
from PIL import Image, ImageChops
import numpy as np
image = Image.open("mozart.gif")
shifted = [bytes(np.roll(row, -row.tolist().index(195)).tolist()
) for row in np.array(image)]
Image.frombytes(image.mode, image.size, b"".join(shifted)).show(
)
```

Solution 2

```
from PIL import Image, ImageChops

image = Image.open("mozart.gif")

for y in range(image.size[1]):
    box = 0, y, image.size[0], y + 1
    row = image.crop(box)
    bytes = row.tobytes()
    i = bytes.index(195)
    row = ImageChops.offset(row, -i)
    image.paste(row, box)

image.save("level16-result.gif")
```

Result: romance!

Next Level

http://www.pythonchallenge.com/pc/return/romance.html

Python Challenge - Level 17

Problem Link

- Link: http://www.pythonchallenge.com/pc/return/romance.html
- Username: **huge**
- Password: **file**

Clues

Clue 1

Image: it is named **cookies**

Clue 2

There's an embedded image. Looks familiar? It's from Level 4.

Solution

Part 1:

```
from urllib.request import urlopen
from urllib.parse import unquote_plus, unquote_to_bytes
import re, bz2

num = '12345'
info = ''
while(True):
    h = urlopen('http://www.pythonchallenge.com/pc/def/linkedlis
t.php?busynothing='+num)
    raw = h.read().decode("utf-8")
    print(raw)
    cookie = h.getheader("Set-Cookie")
    info += re.search('info=(.*?);', cookie).group(1)
    match = re.search('the next busynothing is (\d+)', raw)
    if match == None:
        break
    else:
        num = match.group(1)
res = unquote_to_bytes(info.replace("+", " "))
print(res)
print(bz2.decompress(res).decode())
```

Output:

```
...
and the next busynothing is 96070
and the next busynothing is 83051
that's it.
b'BZh91AY&SY\x94:\xe2I\x00\x00!\x19\x80P\x81\x11\x00\xafg\x9e\xa
0 \x00hE=M\xb5#\xd0\xd4\xd1\xe2\x8d\x06\xa9\xfa&S\xd4\xd3!\xa1\x
eai7h\x9b\x9a+\xbf`"\xc5WX\xe1\xadL\x80\xe8V<\xc6\xa8\xdbH&32\x1
8\xa8x\x01\x08!\x8dS\x0b\xc8\xaf\x96KO\xca2\xb0\xf1\xbd\x1du\xa0
\x86\x05\x92s\xb0\x92\xc4Bc\xf1w$S\x85\t\tC\xae$\x90'
```

And the final line:

```
is it the 26th already? call his father and inform him that "the
 flowers are on their way". he'll understand.
```

Part 2:

```
from xmlrpc.client import ServerProxy

conn = ServerProxy("http://www.pythonchallenge.com/pc/phonebook.
php")

print(conn.phone("Leopold"))
```

```
555-VIOLIN
```

Visit http://www.pythonchallenge.com/pc/return/violin.html

```
no! i mean yes! but ../stuff/violin.php.
```

Part 3:

```
from urllib.request import Request, urlopen
from urllib.parse import quote_plus

url = "http://www.pythonchallenge.com/pc/stuff/violin.php"
msg = "the flowers are on their way"
req = Request(url, headers = { "Cookie": "info=" + quote_plus(ms
g)})

print(urlopen(req).read().decode())
```

```
<html>
<head>
  <title>it's me. what do you want?</title>
  <link rel="stylesheet" type="text/css" href="../style.css">
</head>
<body>
    <br><br>
    <center><font color="gold">
    <img src="leopold.jpg" border="0"/>
<br><br>
oh well, don't you dare to forget the balloons.</font>
</body>
</html>
```

Next Level

http://www.pythonchallenge.com/pc/stuff/balloons.html

redirect to

http://www.pythonchallenge.com/pc/return/balloons.html

Python Challenge - Level 18

- Link: http://www.pythonchallenge.com/pc/return/balloons.html
- Username: **huge**
- Password: **file**

Problem

Solution

The difference is "brightness"... so go to

http://www.pythonchallenge.com/pc/return/brightness.html

In HTML source:

```
<!-- maybe consider deltas.gz -->
```

Download: http://www.pythonchallenge.com/pc/return/deltas.gz

```
>>> print(gzip.open("deltas.gz").read().decode())
```

You will see the data consists roughly two columns.

```
import gzip, difflib

data = gzip.open("deltas.gz")
d1, d2 = [], []
for line in data:
    d1.append(line[:53].decode()+"\n")
    d2.append(line[56:].decode())


compare = difflib.Differ().compare(d1, d2)

f = open("f.png", "wb")
f1 = open("f1.png", "wb")
f2 = open("f2.png", "wb")

for line in compare:
    bs = bytes([int(o, 16) for o in line[2:].strip().split(" ")
if o])
    if line[0] == '+':
        f1.write(bs)
    elif line[0] == '-':
        f2.write(bs)
    else:
        f.write(bs)

f.close()
f1.close()
f2.close()
```

`difflib.Differ().compare(a, b)` will generate

- lines start with `+` : appear in a not in b
- lines start with `-` : appear in b not in a
- others: appear in both

Result:

- `f.png` : ../hex/bin.html

- `f1.png` : butter
- `f2.png` : fly

Next Level

http://www.pythonchallenge.com/pc/hex/bin.html

- username: butter
- password: fly

Python Challenge - Level 19

- Link: http://www.pythonchallenge.com/pc/hex/bin.html
- Username: **butter**
- Password: **fly**

Problem

Solution

```
import email, base64, wave

message = open("email.txt", "rb").read().decode()

mail = email.message_from_string(message)

audio = mail.get_payload(0).get_payload(decode=True)

f = open("indian.wav", "wb")

f.write(audio)



w = wave.open('indian.wav', 'rb')

h = wave.open("result.wav", "wb")

print(w.getnchannels())
print(w.getsampwidth())
print(w.getframerate())
h.setnchannels(w.getnchannels())
h.setsampwidth(w.getsampwidth()//2)
h.setframerate(w.getframerate()*2)
frames = w.readframes(w.getnframes())
wave.big_endiana = 1
h.writeframes(frames)

h.close()
```

Next Level

http://www.pythonchallenge.com/pc/hex/idiot2.html

Python Challenge - Level 20

- Link: http://www.pythonchallenge.com/pc/hex/idiot2.html
- Username: **butter**
- Password: **fly**

Problem

> but inspecting it carefully is allowed.

Solution

```
import urllib.request, base64

request = urllib.request.Request('http://www.pythonchallenge.com
/pc/hex/unreal.jpg')
cred = base64.b64encode(b"butter:fly")
request.add_header("Authorization", "Basic %s" % cred.decode())
print(request.headers)
# {'Authorization': 'Basic YnV0dGVyOmZseQ=='}

response = urllib.request.urlopen(request)
print(response.headers)
# Content-Type: image/jpeg
# Content-Range: bytes 0-30202/2123456789
# Connection: close
# Transfer-Encoding: chunked
# Server: lighttpd/1.4.35
```

So it is a huge file(length of 2123456789) however only a small portion is served(0 to 30202). Try to get the content after that:

```
pattern = re.compile('bytes (\d+)-(\d+)/(\d+)')
content_range = response.headers['content-range']
(start, end, length) = pattern.search(content_range).groups()
request.headers['Range'] = 'bytes=%i-' % (int(end) + 1)
response = urllib.request.urlopen(request)
print(response.headers)
# {'Authorization': 'Basic YnV0dGVyOmZseQ=='}
# Content-Type: application/octet-stream
# Content-Transfer-Encoding: binary
# Content-Range: bytes 30203-30236/2123456789
# Connection: close
# Transfer-Encoding: chunked
# Server: lighttpd/1.4.35

print(response.read().decode())
# Why don't you respect my privacy?
```

So now the content between 30203 and 30236 is served, which is "Why don't you respect my privacy?"; continue for a few iterations:

```
pattern = re.compile('bytes (\d+)-(\d+)/(\d+)')
content_range = response.headers['content-range']
(start, end, length) = pattern.search(content_range).groups()

while True:
    try:
        request.headers['Range'] = 'bytes=%i-' % (int(end) + 1)
        response = urllib.request.urlopen(request)
        print(response.read().decode())
        print(response.headers)
        (start, end, length) = pattern.search(response.headers['
content-range']).groups()
    except:
        break
```

It prints:

```
Why don't you respect my privacy?

we can go on in this way for really long time.

stop this!

invader! invader!

ok, invader. you are inside now.
```

The last request ends at 30346.

Go to http://www.pythonchallenge.com/pc/hex/invader.html

> Yes! that's you!

What about content after the length:

```
request.headers['Range'] = 'bytes=%i-' % (int(length) + 1)
response = urllib.request.urlopen(request)
print(response.headers)
print(response.read().decode())
```

Result:

```
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-Range: bytes 2123456744-2123456788/2123456789
Connection: close
Transfer-Encoding: chunked
Server: lighttpd/1.4.35

esrever ni emankcin wen ruoy si drowssap eht
```

The content is reversed: "the password is your new nickname in reverse". The "nickname" is "invader", so password is "redavni".

Now "reverse" the search:

```
request.headers['Range'] = 'bytes=2123456743-'
response = urllib.request.urlopen(request)
print(response.headers)
print(response.read().decode())
```

```
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-Range: bytes 2123456712-2123456743/2123456789
Connection: close
Transfer-Encoding: chunked
Date: Mon, 02 May 2016 18:12:45 GMT
Server: lighttpd/1.4.35


and it is hiding at 1152983631.
```

Then save it as a zip file:

```
request.headers['Range'] = 'bytes=1152983631-'
response = urllib.request.urlopen(request)

with open("level21.zip", "wb") as f:
    f.write(response.read())
```

Unzip it with the password("redavni").

> Yes! This is really level 21 in here. And yes, After you solve it, you'll be in level 22!
>
> Now for the level:
>
> - We used to play this game when we were kids
> - When I had no idea what to do, I looked backwards.

Next Level

Inside the zip file

Python Challenge - Level 21

Problem

Inside the zip file.

Solution

```
import zlib, bz2

with open("package.pack", "rb") as f:
    data = f.read()

    while True:
        if data.startswith(b'x\x9c'):
            data = zlib.decompress(data)
        elif data.startswith(b'BZh'):
            data = bz2.decompress(data)
        elif data.endswith(b'\x9cx'):
            data = data[::-1]
        else:
            break
    print(data)
```

Result:

```
b'sgol ruoy ta kool'
```

Add logging:

```
import zlib, bz2

result = ""

with open("package.pack", "rb") as f:
    data = f.read()

    while True:
        if data.startswith(b'x\x9c'):
            data = zlib.decompress(data)
            result += ' '
        elif data.startswith(b'BZh'):
            data = bz2.decompress(data)
            result += '#'
        elif data.endswith(b'\x9cx'):
            data = data[::-1]
            result += '\n'
        else:
            break
    print(result)
```

```
      ###           ###       ########    ########    ##########
########
    #######       #######    #########   #########   #########
#########
   ##     ##     ##     ##   ##       ##  ##       ##  ##
##       ##
  ##           ##         ##  ##       ##  ##       ##  ##
##       ##
  ##           ##         ##  #########   #########   ########
#########
  ##           ##         ##  ########    ########    ########
########
  ##           ##         ##  ##          ##          ##
##   ##
   ##     ##     ##     ##   ##          ##          ##
##     ##
    #######       #######    ##          ##          #########
##      ##
      ###           ###       ##          ##          ##########
##       ##
```

Next Level

http://www.pythonchallenge.com/pc/hex/copper.html

Python Challenge - Level 22

- Link: http://www.pythonchallenge.com/pc/hex/copper.html
- Username: **butter**
- Password: **fly**

Problem

Solution

```
from PIL import Image, ImageDraw

img = Image.open("white.gif")
new = Image.new("RGB", (500, 200))
draw = ImageDraw.Draw(new)
cx, cy = 0, 100
for frame in range(img.n_frames):
    img.seek(frame)
    left, upper, right, lower = img.getbbox()

    # get the direction; like a joystick,
    dx = left - 100
    dy = upper - 100

    # end of a move(letter), shift to the right
    if dx == dy == 0:
        cx += 50
        cy = 100
    cx += dx
    cy += dy
    draw.point([cx, cy])

new.show()
```

Result: bonus

Next Level

http://www.pythonchallenge.com/pc/hex/bonus.html

Python Challenge - Level 23

- Link: http://www.pythonchallenge.com/pc/hex/bonus.html
- Username: **butter**
- Password: **fly**

Problem

Solution

`this` is a special module in Python:

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to d
o it.
Although that way may not be obvious at first unless you're Dutc
h.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

The string is internally stored as `this.s`:

```
>>> print(this.s)
Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb q
b vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgp
u.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!
```

The text is `rot-13` encoded(rotate by 13 characters.

```
>>> import this
>>> print(codecs.decode(this.s, "rot-13"))
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to d
o it.
Although that way may not be obvious at first unless you're Dutc
h.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

And there's a built-in dict for the rotation:

```
>>> this.d
{'n': 'a', 'J': 'W', 'o': 'b', 'r': 'e', 'T': 'G', 'K': 'X', 'v'
: 'i', 'H': 'U', 'L': 'Y', 'q': 'd', 'w': 'j', 'O': 'B', 'e': 'r'
, 'S': 'F', 'l': 'y', 'X': 'K', 'I': 'V', 'a': 'n', 'Z': 'M', 'G'
: 'T', 'i': 'v', 'f': 's', 'u': 'h', 'h': 'u', 'N': 'A', 'R': 'E'
, 'P': 'C', 'M': 'Z', 'C': 'P', 'b': 'o', 'U': 'H', 'A': 'N', 'B'
: 'O', 'W': 'J', 'D': 'Q', 'k': 'x', 'c': 'p', 'x': 'k', 'm': 'z'
, 'F': 'S', 'V': 'I', 'E': 'R', 'Q': 'D', 'p': 'c', 'g': 't', 'Y'
: 'L', 'j': 'w', 'y': 'l', 's': 'f', 't': 'g', 'z': 'm', 'd': 'q'
}
```

So you can also do:

```
>>> print("".join([this.d.get(c, c) for c in this.s]))
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to d
o it.
Although that way may not be obvious at first unless you're Dutc
h.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Use this to decode the message:

```
>>> s = 'va gur snpr bs jung?'
>>> print("".join([this.d.get(c, c) for c in s]))
in the face of what?
```

From the "Zen":

> In the face of ambiguity, ...

The answer is "ambiguity"

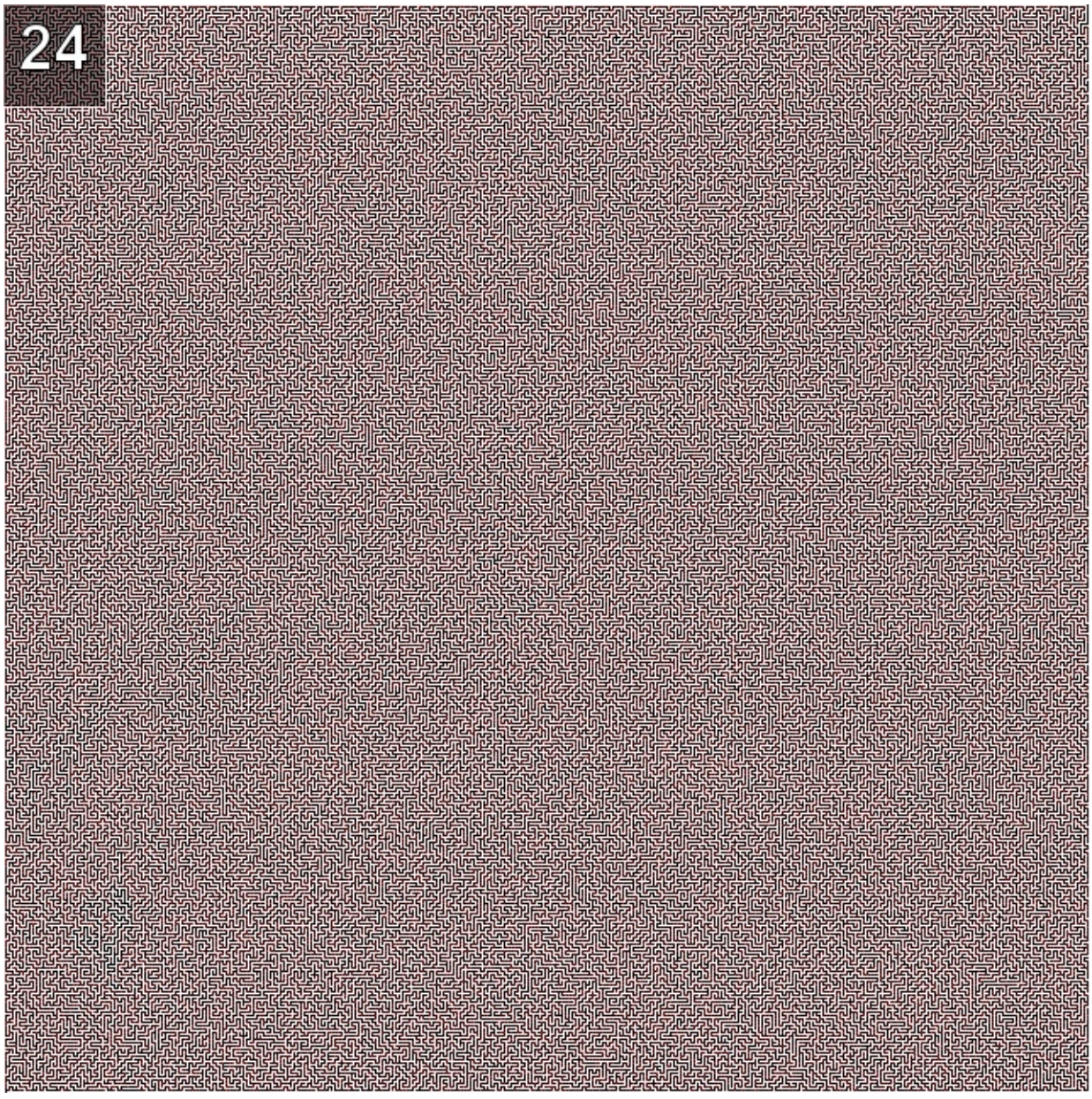
Next Level

http://www.pythonchallenge.com/pc/hex/ambiguity.html

Python Challenge - Level 24

- Link: http://www.pythonchallenge.com/pc/hex/ambiguity.html
- Username: **butter**
- Password: **fly**

Problem



Solution

Load the image:

```
>>> from PIL import Image
>>> maze = Image.open("maze.png")
>>> w, h = maze.size
```

Check the top line

```
>>> for i in range(w): print(maze.getpixel((i, 0)))
```

There's only one black pixel(`(0, 0, 0, 255)`), others are white(`(255, 255, 255, 255)`) or grey(the upper left corner square with the number of the level), so the entrance point is `(w - 2, 0)` .

Similarly print out the bottom line:

```
>>> for i in range(w): print(maze.getpixel((i, h - 1)))
```

there's only one black pixel at `(1, h - 1)` , that would be the exit point.

By printing out the "inner" pixels you may notice that the non-white points all look like `(x, 0, 0, 255)` , where `x` would vary. That is the data we need to collect.

A BFS would find the shortest path:

```
from PIL import Image

maze = Image.open("maze.png")
directions = [(0,1), (0,-1), (1,0), (-1,0)]
white = (255, 255, 255, 255)
w, h = maze.size

next_map = {}

entrance = (w - 2, 0)
exit = (1, h - 1)
queue = [exit]
while queue:
    pos = queue.pop(0)
    if pos == entrance:
        break
    for d in directions:
        tmp = (pos[0] + d[0], pos[1] + d[1])
        if not tmp in next_map and 0 <= tmp[0] < w and 0 <= tmp[1
] < h and maze.getpixel(tmp) != white:
            next_map[tmp] = pos
            queue.append(tmp)

path = []
while pos != exit:
    path.append(maze.getpixel(pos)[0])
    pos = next_map[pos]

# skipping the 0s
print(path[1::2])
open('maze.zip','wb').write(bytes(path[1::2]))
```

From the zip, a picture with the word "lake".

Next Level

http://www.pythonchallenge.com/pc/hex/lake.html

Python Challenge - Level 25

- Link: http://www.pythonchallenge.com/pc/hex/lake.html
- Username: **butter**
- Password: **fly**

Problem

Solution

To download a wav file, use wget with authentication:

$ wget --user butter --password fly
http://www.pythonchallenge.com/pc/hex/lake1.wav

To download all the wav files:

```
$ for i in {1..25}; do wget --user butter --password fly http://
www.pythonchallenge.com/pc/hex/lake$i.wav; done
```

25 wav files can be converted to 5 by 5 = 25 pieces of images

```
from PIL import Image
import wave

wavs = [wave.open('lake/lake%d.wav' % i) for i in range(1,26)]
result = Image.new('RGB', (300,300), 0)
num_frames = wavs[0].getnframes()
for i in range(len(wavs)):
    byte = wavs[i].readframes(num_frames)
    img = Image.frombytes('RGB', (60, 60), byte)
    result.paste(img, (60 * (i % 5), 60 * (i // 5)))
result.save('level25.png')
```

Next Level

www.pythonchallenge.com/pc/hex/decent.html

Python Challenge - Level 26

- Link: www.pythonchallenge.com/pc/hex/decent.html
- Username: **butter**
- Password: **fly**

Problem

> Hurry up, I'm missing the boat

Solution

Send the email to `leopold.moz@pythonchallenge.com` and get the response:

> From: leopold.moz@pythonchallenge.com Subject: Re: sorry Date:
>
> Never mind that.
>
> Have you found my broken zip?
>
> md5: bbb8b499a0eef99b52c7f13f4e78c24b
>
> Can you believe what one mistake can lead to?

As indicated there's only "one mistake". Try to modify the data and check by md5 code.

```
import hashlib

def search_and_save():
    for i in range(len(data)):
        for j in range(256):
            newData = data[:i] + bytes([j]) + data[i + 1:]
            if hashlib.md5(newData).hexdigest() == md5code:
                open('repaired.zip', 'wb').write(newData)
                return

md5code = 'bbb8b499a0eef99b52c7f13f4e78c24b'
data = open('maze/mybroken.zip', 'rb').read()
search_and_save()
```

The result is a picture with word "speed", plus the text:

> Hurry up, I'm missing the boat

The final answer is **speedboat**

Next Level

http://www.pythonchallenge.com/pc/hex/speedboat.html

Python 2 to 3

- `md5` module is deprecated, use `hashlib.md5()`

- `open()` without explicitly specifying `rb` will use the default `utf-8` codec for decoding.

Python Challenge - Level 27

- Link: http://www.pythonchallenge.com/pc/hex/speedboat.html
- Username: **butter**
- Password: **fly**

Problem

Solution

```
>>> palette = im.getpalette()[::3]
>>> len(palette)
256

>>> b = bytes([i for i in range(256)])
>>> b
b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10
\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#
$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`a
bcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x8
7\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x9
7\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa
7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb
7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc
7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd
7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe
7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf
7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff'
>>> len(b)
256
```

Full Solution

```
from PIL import Image
import bz2

im = Image.open('zigzag.gif')


palette = im.getpalette()[::3]

table = bytes.maketrans(bytes([i for i in range(256)]), bytes(pa
lette))

raw = im.tobytes()

trans = raw.translate(table)

zipped = list(zip(raw[1:], trans[:-1]))

diff = list(filter(lambda p: p[0] != p[1], zipped))

indices = [i for i,p in enumerate(zipped) if p[0] != p[1]]

im2 = Image.new("RGB", im.size)
colors = [(255, 255, 255)] * len(raw)
for i in indices:
    colors[i] = (0, 0, 0)
im2.putdata(colors)
#im2.show()


s = [t[0] for t in diff]
text = bz2.decompress(bytes(s))

import keyword
print(set([w for w in text.split() if not keyword.iskeyword(w.de
code())]))
```

Result:

```
{b'exec', b'print', b'../ring/bell.html', b'repeat', b'switch'}
```

Well, this level needs to be updated, since `exec` and `print` are not keywords in Python 3. The only two words left are `repeat` and `switch`

Next Level

http://www.pythonchallenge.com/pc/ring/bell.html

- username: repeat
- password: switch

Python Challenge - Level 28

- Link: http://www.pythonchallenge.com/pc/ring/bell.html
- Username: **repeat**
- Password: **switch**

Problem

> RING-RING-RING say it out loud

Solution

http://www.pythonchallenge.com/pc/ring/green.html

> yes! green!

```
from PIL import Image

im = Image.open('bell.png')

# split RGB and get Green
green = list(im.split()[1].getdata())

# calculate diff for every two bytes
diff = [abs(a - b) for a, b in zip(green[0::2], green[1::2])]

# remove the most frequent value 42
filtered = list(filter(lambda x: x != 42, diff))

# convert to string and print out
print(bytes(filtered).decode())
```

Result

```
whodunnit().split()[0] ?
```

The creator of Python is Guido van Rossum, so the final answer is "guido"

Next Level

http://www.pythonchallenge.com/pc/ring/guido.html

Python Challenge - Level 29

- Link: http://www.pythonchallenge.com/pc/ring/guido.html
- Username: **repeat**
- Password: **switch**

Problem

Solution

Notice that there "silent" empty lines in the html source code. And the lines are of different lengths. Convert the lengths to bytes and decompress by bz2:

```
from urllib.request import Request, urlopen
import bz2, base64

req = Request('http://www.pythonchallenge.com/pc/ring/guido.html'
)
req.add_header('Authorization', 'Basic %s' % base64.b64encode(b'
repeat:switch').decode())
raw = urlopen(req).read().splitlines()[12:]
data = bytes([len(i) for i in raw])
print(bz2.decompress(data))
```

```
b"Isn't it clear? I am yankeedoodle!"
```

Next Level

http://www.pythonchallenge.com/pc/ring/yankeedoodle.html

Python Challenge - Level 30

- Link: http://www.pythonchallenge.com/pc/ring/yankeedoodle.html
- Username: **repeat**
- Password: **switch**

Problem

> The picture is only meant to help you relax

Solution

$ wget --user repeat --password switch
http://www.pythonchallenge.com/pc/ring/yankeedoodle.csv

```
from PIL import Image
import math

with open('yankeedoodle.csv') as f:
    data = [x.strip() for x in f.read().split(",")]
    length = len(data)
    print(length)
    # 7367

    factors = [x for x in range(2, length) if length % x == 0]
    print(factors)
    # [53, 139]

    img = Image.new("F", (53, 139))
    img.putdata([float(x) for x in data], 256)

    img = img.transpose(Image.FLIP_LEFT_RIGHT)
    img = img.transpose(Image.ROTATE_90)
    #img.show()

    a = data[0::3]
    b = data[1::3]
    c = data[2::3]

    res = bytes([int(x[0][5] + x[1][5] + x[2][6]) for x in zip(d
ata[0::3], data[1::3], data[2::3])])

    print(res)
```

```
b'So, you found the hidden message.\nThere is lots of room here
for a long message, but we only need very little space to say "l
ook at grandpa", so the rest is just garbage. \nVTZ.l\'\x7ftf*Om
@I"p]...'
```

keyword: grandpa

Next Level

http://www.pythonchallenge.com/pc/ring/grandpa.html

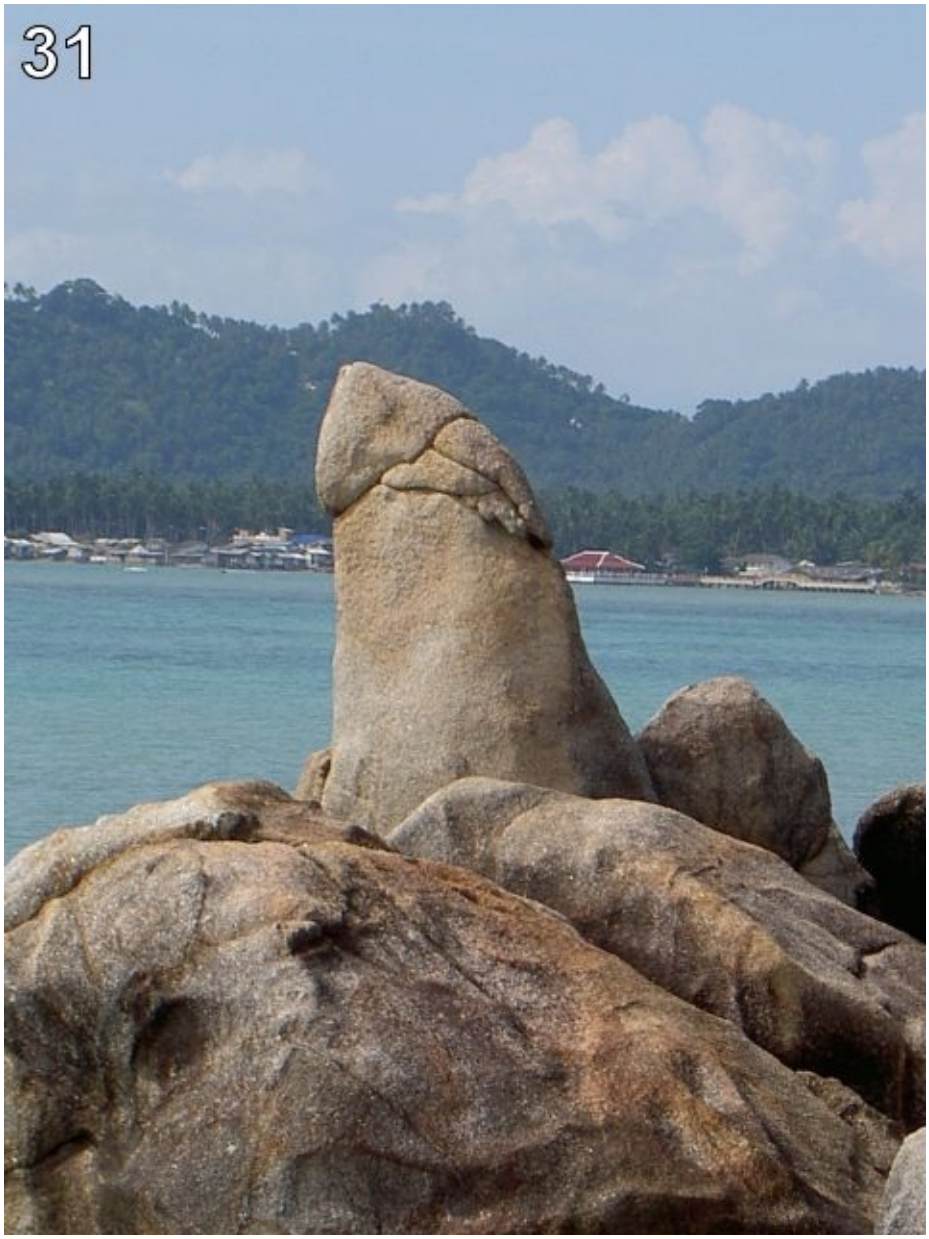
Python Challenge - Level 31

- Link: http://www.pythonchallenge.com/pc/ring/grandpa.html
- Username: **repeat**
- Password: **switch**

Click image

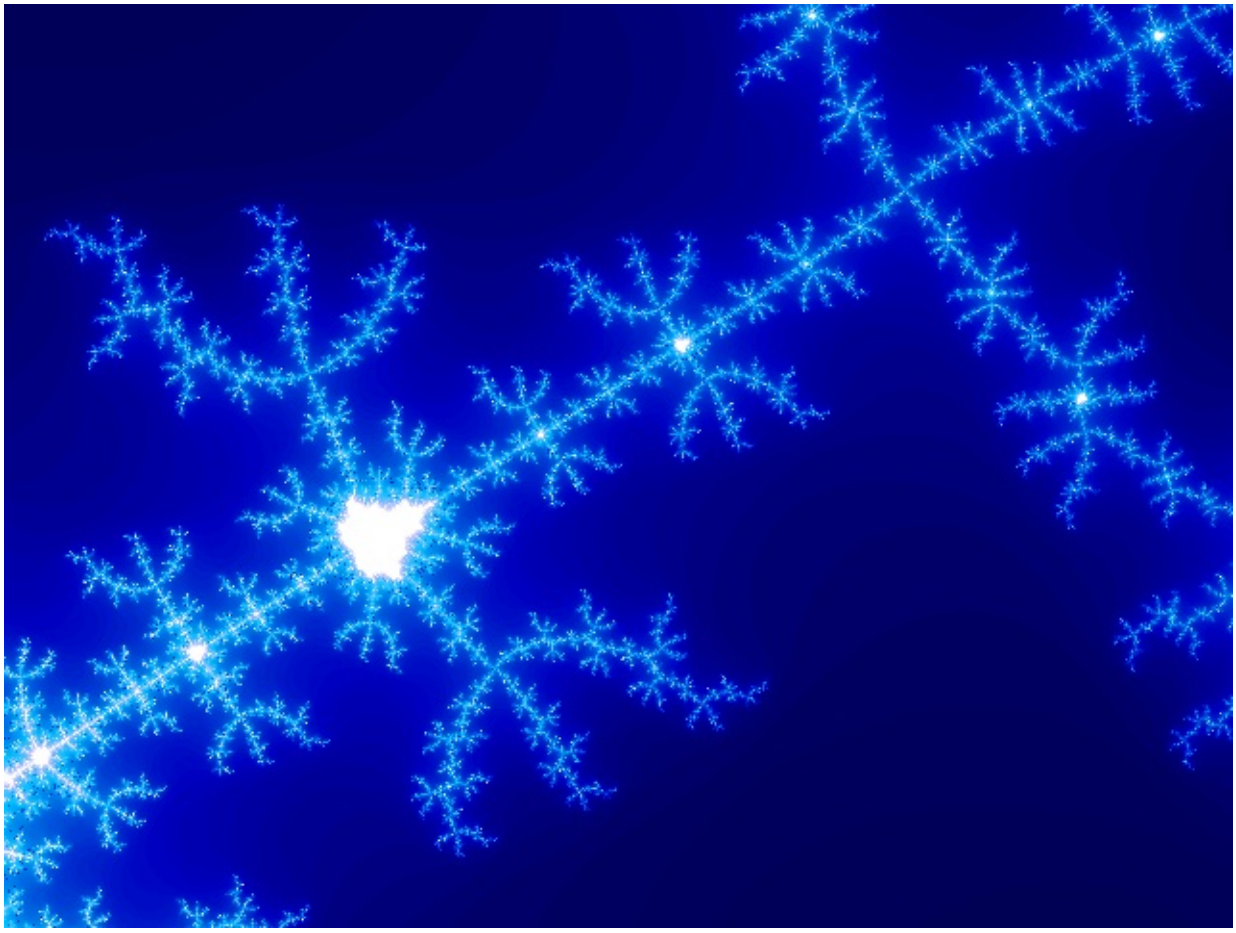
- Username: **kohsamui**
- Password: **thailand**

Problem



Click on the image:

> That was too easy. You are still on 31...



In page source:

```
<img src="mandelbrot.gif" border="0">
<window left="0.34" top="0.57" width="0.036" height="0.027"/>
<option iterations="128"/>
```

Solution

It is a image of the Mandelbrot set.

> The Mandelbrot set is the set of complex numbers c for which the function f(z) = z^2 + c does not diverge when iterated from z = 0, i.e., for which the sequence f(0), f(f(0)), etc., remains bounded in absolute value.

To slightly translate this into English: every pixel in the 2-d image `(x, y)`, uniquely represents a complex number `c`, we let `z = 0 + 0j`, then repeatedly calculate `z = z * z + c`, if `z` is always within a range(not going to infinity),

we light this pixel up, otherwise we mark it as dark. Actually we check at which iteration it starts to diverge, and set color to that number of iteration.

In page source we are given 4 number: left, top, width, height, that is the bound of our calculation. Suppose the image we are going to produce has width `w` and height `h`, then `x` should be from `0` to `w - 1`, and `y` should be from `0` to `h - 1`. The real part of the `c` can be computed by `left + x * width / w`, and imaginary part is `bottom + y * height / h`

```
from PIL import Image

img = Image.open("mandelbrot.gif")

left = 0.34
bottom = 0.57
width = 0.036
height = 0.027
max = 128

w, h = img.size

xstep = width / w
ystep = height/ h

result = []

for y in range(h - 1, -1, -1):
    for x in range(w):
        c = complex(left + x * xstep, bottom + y * ystep)
        z = 0 + 0j
        for i in range(max):
            z = z * z + c
            if abs(z) > 2:
                break
        result.append(i)

img2 = img.copy()
img2.putdata(result)
img2.show()
```

```
diff = [(a - b) for a, b in zip(img.getdata(), img2.getdata()) if
 a != b]
print(len(diff))

plot = Image.new('L', (23, 73))
plot.putdata([(i < 16) and 255 or 0 for i in diff])
plot.resize((230,730)).show()
```

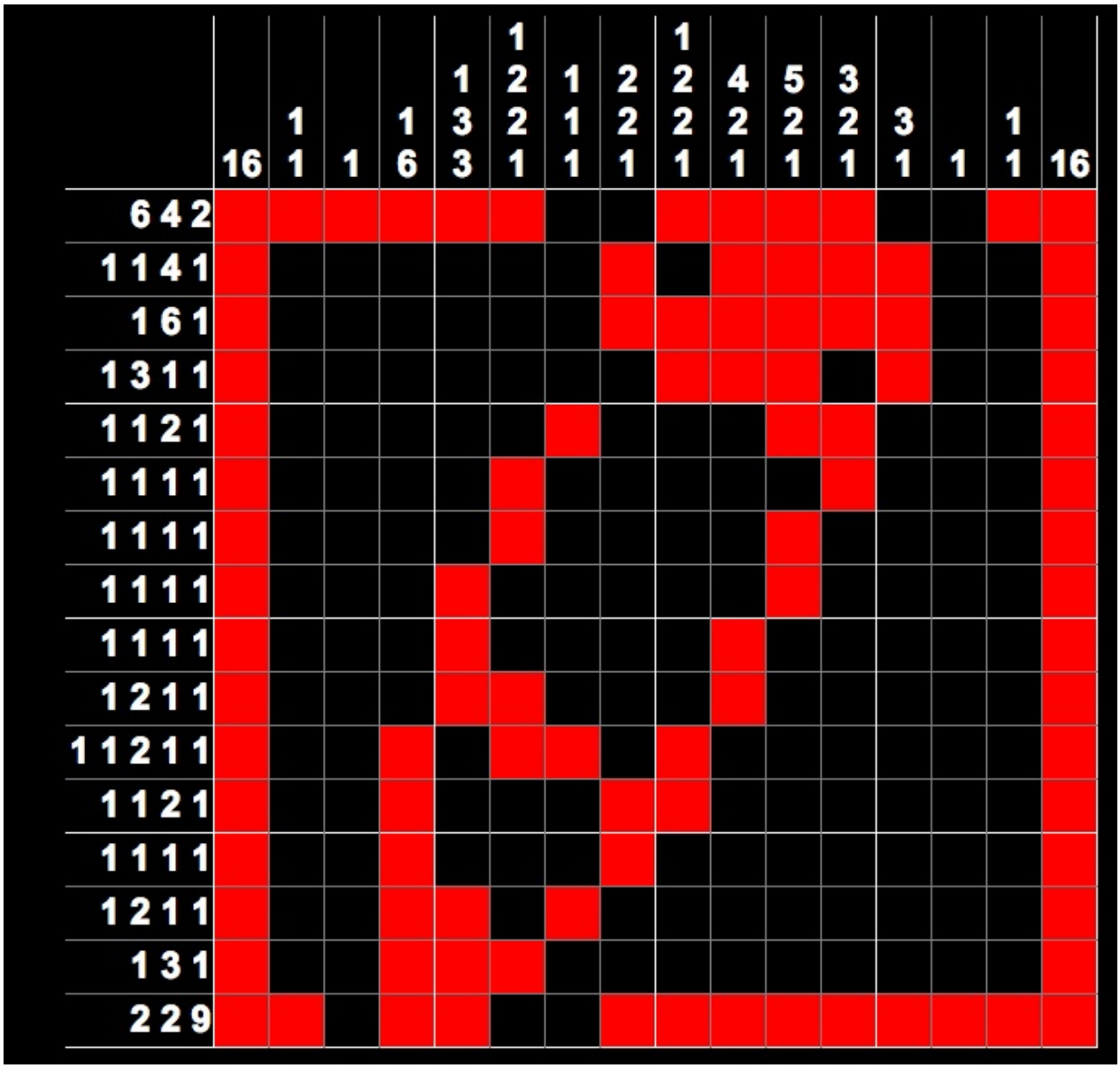
Extra Links

https://en.wikipedia.org/wiki/Arecibo_message

Python Challenge - Level 32

- Link: http://www.pythonchallenge.com/pc/rock/arecibo.html
- Username: **kohsamui**
- Password: **thailand**

Problem



Solution

```
import time


class Seg:
    def __init__(self, length):
        self.length = length
        self.placements = []

    def __repr__(self):
        return '<Seg:len=' + str(self.length) + ',placements=' +
 str(self.placements) + '>'


class Bar:
    def __init__(self, axis, index, segs, length):
        self.segs = segs
        self.complete = False
        self.dirty = False
        self.axis = axis
        self.index = index
        self.length = length
        self.full_space = sum([seg.length for seg in self.segs])
 + len(self.segs) - 1

    def is_full(self):
        return self.full_space == self.length

    def __repr__(self):
        return '<Bar:axis=' + str(self.axis) + ',index=' + str(s
elf.index) + ',len=' + str(
            self.length) + ',full_space=' + str(
            self.full_space) + ',segs=' + str(self.segs) + ',dir
ty=' + str(self.dirty) + '>'

    def __lt__(self, other):
        return self.index - other.index


def load():
```

```
    with open("up.txt") as f:
        lines = f.readlines()
        lines = [line.strip() for line in lines if (not line.sta
rtswith('#')) and len(line.strip()) != 0]
        raw = [list(map(int, line.split())) for line in lines]
        assert sum(raw[0]) == len(raw) - 1

        return raw[1:raw[0][0] + 1], raw[raw[0][0] + 1:]


def print_board(board):
    m = {1: '#', 0: ' ', None: '?'}
    print("\n".join([''.join([m[ele] for ele in row]) for row in
 board]))


def calc_space(segs):
    return sum([seg.length for seg in segs]) + len(segs) - 1


def calc_placement(bars):
    for bar in bars:
        for i in range(len(bar.segs)):
            seg = bar.segs[i]
            start = calc_space(bar.segs[:i]) + 1
            end = bar.length - calc_space(bar.segs[i + 1:]) - se
g.length
            seg.placements = list(range(start, end))


def update_board(board, bar, i, value):
    if board[bar.axis][bar.index][i] == value:
        return False
    board[bar.axis][bar.index][i] = value
    board[1 - bar.axis][i][bar.index] = value
    return True


def mark_overlaps(board, bars):
    for bar in bars:
```

```
        for seg in bar.segs:
            for i in range(seg.placements[-1], seg.placements[0]
 + seg.length):
                if update_board(board, bar, i, 1):
                    bars[(1 - bar.axis) * h + i].dirty = True


def validate_placement(board, bar, iseg, placement):
    seg = bar.segs[iseg]

    # if the previous pixel is 1
    if placement > 0 and board[bar.axis][bar.index][placement - 1
] == 1:
        return False

    # if any of the pixel inside the segment is 0
    elif 0 in board[bar.axis][bar.index][placement:placement + s
eg.length]:
        return False

    # if the next pixel is 1
    elif placement + seg.length < len(board[bar.axis][bar.index]
) and board[bar.axis][bar.index][
                placement + seg.length] == 1:
        return False

    return True


def mark_flags(board, bar, tmp_placements, flags):
    prev = 0
    for i in range(len(tmp_placements)):
        seg = bar.segs[i]
        placement = tmp_placements[i]
        # print(seg, placement, placement + seg.length)
        for j in range(prev, placement):
            flags[j] |= 2
        for j in range(placement, placement + seg.length):
            flags[j] |= 1
```

```
        end = len(board[bar.axis][0])
        if i != len(tmp_placements) - 1:
            end = tmp_placements[i + 1]

        for j in range(placement + seg.length, end):
            flags[j] |= 2

        prev = placement + seg.length


# flag = 1: can be black,
# flag = 2: can be white,
# flag = 3: can be both
def check_bar(board, bar, start=0, iseg=0, tmp_placements=[], fl
ags=[]):
    if iseg == len(bar.segs):
        last_seg = bar.segs[-1]
        if 1 in board[bar.axis][bar.index][tmp_placements[-1] +
last_seg.length:]:
            return
        mark_flags(board, bar, tmp_placements, flags)
    else:
        seg = bar.segs[iseg]
        valid_placements = []
        for placement in seg.placements:
            # print(validate_placement(board, bar, iseg, placeme
nt))
            if validate_placement(board, bar, iseg, placement):
                valid_placements.append(placement)

        seg.placements = valid_placements

        for placement in seg.placements:
            if placement < start:
                continue

            if 1 in board[bar.axis][bar.index][start:placement]:
                continue

            tmp_placements[iseg] = placement
```

```
            check_bar(board, bar, start=placement + seg.length +
1, iseg=iseg + 1, tmp_placements=tmp_placements,
                      flags=flags)


(hlens, vlens) = load()

h = len(hlens)
v = len(vlens)

bars = []

for ind in range(len(hlens)):
    segs = [Seg(i) for i in hlens[ind]]
    bars.append(Bar(0, ind, segs, h))

for ind in range(len(hlens)):
    segs = [Seg(i) for i in vlens[ind]]
    bars.append(Bar(1, ind, segs, h))

board = [[[None] * v for i in range(h)], [[None] * v for i in ra
nge(h)]]

calc_placement(bars)

mark_overlaps(board, bars)

while True:

    dirty_bars = [(sum([len(seg.placements) for seg in bar.segs]
), bar) for bar in bars if bar.dirty]
    if len(dirty_bars) == 0:
        break
    effort, bar = min(dirty_bars)

    flags = [0] * len(board[bar.axis][0])

    print("Processing Bar: (" + str(bar.axis) + "," + str(bar.in
dex) + ")")
    check_bar(board, bar, tmp_placements=[0] * len(bar.segs), fl
```

```
ags=flags)

    for i in range(len(flags)):
        flag = flags[i]
        if flag == 1:
            if update_board(board, bar, i, 1):
                bars[(1 - bar.axis) * h + i].dirty = True
        elif flag == 2:
            if update_board(board, bar, i, 0):
                bars[(1 - bar.axis) * h + i].dirty = True
    bar.dirty = False

print_board(board[0])

print(time.process_time())
```

Solve warmup.txt, the result is an arrow point up.

Solve up.txt, the result is a snake, try python.html

> Congrats! You made it through to the smiling python.
>
> “Free” as in “Free speech”, not as in “free…

From wikipedia, the word following "free" should be beer.

Next Level

Level 33: http://www.pythonchallenge.com/pc/rock/beer.html

Python Challenge - Level 33

- Link: http://www.pythonchallenge.com/pc/rock/beer.html
- Username: **kohsamui**
- Password: **thailand**

Problem

Solution

```
>>> im.size
(138, 138)
>>> im.mode
'L'
>>> list(im.getdata())
[1, 43, 7, 13, 7, 1, 1, 85, 25, 85,...]
>>> im.histogram()
[0, 1532, 232, 0, 0, 0, 0, 963, 189, 0, 0, 0, 0, 724, 329, 0, 0,
0, 0, 549, 243, 0, 0, 0, 0, 144, 424, 0, 0, 0, 0, 119, 328, 0, 0
```

```
, 0, 0, 126, 339, 0, 0, 0, 0, 126, 357, 0, 0, 0, 0, 107, 225, 0,
0, 0, 0, 79, 609, 0, 0, 0, 0, 181, 356, 0, 0, 0, 0, 70, 298, 0, 0
, 0, 0, 23, 164, 0, 0, 0, 0, 26, 354, 0, 0, 0, 0, 47, 341, 0, 0,
0, 0, 139, 257, 0, 0, 0, 0, 104, 505, 0, 0, 0, 0, 192, 224, 0, 0
, 0, 0, 114, 310, 0, 0, 0, 0, 32, 183, 0, 0, 0, 0, 238, 198, 0, 0
, 0, 0, 117, 327, 0, 0, 0, 0, 110, 342, 0, 0, 0, 0, 118, 342, 0,
0, 0, 0, 145, 323, 0, 0, 0, 0, 152, 324, 0, 0, 0, 0, 161, 323, 0
, 0, 0, 0, 175, 317, 0, 0, 0, 0, 183, 317, 0, 0, 0, 0, 171, 337,
0, 0, 0, 0, 198, 318, 0, 0, 0, 0, 241, 283, 0, 0, 0, 0, 1348, 272
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

>>> list(enumerate(im.histogram()))
[(0, 0), (1, 1532), (2, 232), (3, 0), (4, 0), (5, 0), (6, 0), (7
, 963), (8, 189), (9, 0), (10, 0), (11, 0), (12, 0), (13, 724), (
14, 329), (15, 0), (16, 0), (17, 0), (18, 0), (19, 549), (20, 243
), (21, 0), (22, 0), (23, 0), (24, 0), (25, 144), (26, 424), (27
, 0), (28, 0), (29, 0), (30, 0), (31, 119), (32, 328), (33, 0), (
34, 0), (35, 0), (36, 0), (37, 126), (38, 339), (39, 0), (40, 0)
, (41, 0), (42, 0), (43, 126), (44, 357), (45, 0), (46, 0), (47,
0), (48, 0), (49, 107), (50, 225), (51, 0), (52, 0), (53, 0), (54
, 0), (55, 79), (56, 609), (57, 0), (58, 0), (59, 0), (60, 0), (
61, 181), (62, 356), (63, 0), (64, 0), (65, 0), (66, 0), (67, 70
), (68, 298), (69, 0), (70, 0), (71, 0), (72, 0), (73, 23), (74,
164), (75, 0), (76, 0), (77, 0), (78, 0), (79, 26), (80, 354), (
81, 0), (82, 0), (83, 0), (84, 0), (85, 47), (86, 341), (87, 0),
 (88, 0), (89, 0), (90, 0), (91, 139), (92, 257), (93, 0), (94, 0
), (95, 0), (96, 0), (97, 104), (98, 505), (99, 0), (100, 0), (1
01, 0), (102, 0), (103, 192), (104, 224), (105, 0), (106, 0), (1
07, 0), (108, 0), (109, 114), (110, 310), (111, 0), (112, 0), (1
13, 0), (114, 0), (115, 32), (116, 183), (117, 0), (118, 0), (119
, 0), (120, 0), (121, 238), (122, 198), (123, 0), (124, 0), (125
, 0), (126, 0), (127, 117), (128, 327), (129, 0), (130, 0), (131
, 0), (132, 0), (133, 110), (134, 342), (135, 0), (136, 0), (137
, 0), (138, 0), (139, 118), (140, 342), (141, 0), (142, 0), (143
, 0), (144, 0), (145, 145), (146, 323), (147, 0), (148, 0), (149
, 0), (150, 0), (151, 152), (152, 324), (153, 0), (154, 0), (155
, 0), (156, 0), (157, 161), (158, 323), (159, 0), (160, 0), (161
, 0), (162, 0), (163, 175), (164, 317), (165, 0), (166, 0), (167
```

```
, 0), (168, 0), (169, 183), (170, 317), (171, 0), (172, 0), (173
, 0), (174, 0), (175, 171), (176, 337), (177, 0), (178, 0), (179
, 0), (180, 0), (181, 198), (182, 318), (183, 0), (184, 0), (185
, 0), (186, 0), (187, 241), (188, 283), (189, 0), (190, 0), (191
, 0), (192, 0), (193, 1348), (194, 272), (195, 0), (196, 0), (197
, 0), (198, 0), (199, 0), (200, 0), (201, 0), (202, 0), (203, 0)
, (204, 0), (205, 0), (206, 0), (207, 0), (208, 0), (209, 0), (2
10, 0), (211, 0), (212, 0), (213, 0), (214, 0), (215, 0), (216, 0
), (217, 0), (218, 0), (219, 0), (220, 0), (221, 0), (222, 0), (
223, 0), (224, 0), (225, 0), (226, 0), (227, 0), (228, 0), (229,
0), (230, 0), (231, 0), (232, 0), (233, 0), (234, 0), (235, 0), (
236, 0), (237, 0), (238, 0), (239, 0), (240, 0), (241, 0), (242,
0), (243, 0), (244, 0), (245, 0), (246, 0), (247, 0), (248, 0), (
249, 0), (250, 0), (251, 0), (252, 0), (253, 0), (254, 0), (255,
0)]

>>> sum([x for x in im.histogram() if x != 0][:-2])
17424
```

```
from PIL import Image
import math

im = Image.open('beer2.png')

data = list(im.getdata())

out = None

for i in range(33):
    max_value = max(data)
    data = [x for x in data if x < max_value - 1]
    print(len(data))
    l = int(math.sqrt(len(data)))
    out = Image.new('L', (l, l))
    out.putdata(data)
    out.show()
```

The letters with frames:

> gremlins

Final Link

http://www.pythonchallenge.com/pc/rock/gremlins.html

Links

Documentation

- The Python Standard Library: (https://docs.python.org/3/library/index.html)
- PEP 8: Style Guide for Python Code(https://www.python.org/dev/peps/pep-0008/)
- PIL: Python Imaging Library(fork that works with Python 3.x) (http://pillow.readthedocs.io/)

Problems

- Level 0:
  - problem: http://www.pythonchallenge.com/pc/def/0.html
  - unlock official solution: http://www.pythonchallenge.com/pcc/def/map.html
- Level 1:
  - problem: http://www.pythonchallenge.com/pc/def/map.html
  - unlock official solution: http://www.pythonchallenge.com/pcc/def/ocr.html
- Level 2:
  - problem: http://www.pythonchallenge.com/pc/def/ocr.html
  - unlock official solution: http://www.pythonchallenge.com/pcc/def/equality.html
- Level 3:
  - problem: http://www.pythonchallenge.com/pc/def/equality.html
  - unlock official solution: http://www.pythonchallenge.com/pcc/def/linkedlist.php
- Level 4:
  - problem: http://www.pythonchallenge.com/pc/def/linkedlist.php
  - unlock official solution: http://www.pythonchallenge.com/pcc/def/peak.html
- Level 5:
  - problem: http://www.pythonchallenge.com/pc/def/peak.html

    - unlock official solution: http://www.pythonchallenge.com/pcc/def/channel.html
- Level 6:
    - problem: http://www.pythonchallenge.com/pc/def/channel.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/def/oxygen.html
- Level 7:
    - problem: http://www.pythonchallenge.com/pc/def/oxygen.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/def/integrity.html
- Level 8:
    - problem: http://www.pythonchallenge.com/pc/def/integrity.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/good.html
- Level 9:
    - problem: http://www.pythonchallenge.com/pc/return/good.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/bull.html
    - username: huge
    - password: file
- Level 10:
    - problem: http://www.pythonchallenge.com/pc/return/bull.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/5808.html
    - username: huge
    - password: file
- Level 11:
    - problem: http://www.pythonchallenge.com/pc/return/5808.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/evil.html
    - username: huge
    - password: file
- Level 12:
    - problem: http://www.pythonchallenge.com/pc/return/evil.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/disproportional.html

    - username: huge
    - password: file
- Level 13:
    - problem: http://www.pythonchallenge.com/pc/return/disproportional.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/italy.html
    - username: huge
    - password: file
- Level 14:
    - problem: http://www.pythonchallenge.com/pc/return/italy.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/uzi.html
    - username: huge
    - password: file
- Level 15:
    - problem: http://www.pythonchallenge.com/pc/return/uzi.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/mozart.html
    - username: huge
    - password: file
- Level 16:
    - problem: http://www.pythonchallenge.com/pc/return/mozart.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/return/romance.html
    - username: huge
    - password: file
- Level 17:
    - problem: http://www.pythonchallenge.com/pc/return/romance.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/stuff/balloons.html
    - username: huge
    - password: file
- Level 18:
    - problem: http://www.pythonchallenge.com/pc/stuff/balloons.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/bin.html:butter:fly

    - username: huge
    - password: file
- Level 19:
    - problem: http://www.pythonchallenge.com/pc/hex/bin.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/idiot2.html
    - username: butter
    - password: fly
- Level 20:
    - problem: http://www.pythonchallenge.com/pc/hex/idiot2.html
    - username: butter
    - password: fly
- Level 21: NO LINK(inside the zip file from Level 20)
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/copper.html
- Level 22:
    - problem: http://www.pythonchallenge.com/pc/hex/copper.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/bonus.html
    - username: butter
    - password: fly
- Level 23:
    - problem: http://www.pythonchallenge.com/pc/hex/bonus.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/ambiguity.html
    - username: butter
    - password: fly
- Level 24:
    - problem: http://www.pythonchallenge.com/pc/hex/ambiguity.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/lake.html
    - username: butter
    - password: fly
- Level 25:
    - problem: http://www.pythonchallenge.com/pc/hex/lake.html
    - unlock official solution:

    - http://www.pythonchallenge.com/pcc/hex/decent.html
    - username: butter
    - password: fly
- Level 26:
    - problem: http://www.pythonchallenge.com/pc/hex/decent.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/hex/speedboat.html
    - username: butter
    - password: fly
- Level 27:
    - problem: http://www.pythonchallenge.com/pc/hex/speedboat.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/ring/bell.html
    - username: butter
    - password: fly
- Level 28:
    - problem: http://www.pythonchallenge.com/pc/ring/bell.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/ring/guido.html
    - username: repeat
    - password: switch
- Level 29: -problem: http://www.pythonchallenge.com/pc/ring/guido.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/ring/yankeedoodle.html
    - username: repeat
    - password: switch
- Level 30:
    - problem: http://www.pythonchallenge.com/pc/ring/yankeedoodle.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/ring/grandpa.html
    - username: repeat
    - password: switch
- Level 31:
    - problem: http://www.pythonchallenge.com/pc/ring/grandpa.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/rock/arecibo.html
    - username: repeat

    - password: switch
    - username(after clicking the image): kohsamui
    - password(after clicking the image): thailand
- Level 32:
    - problem: http://www.pythonchallenge.com/pc/rock/arecibo.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/rock/beer.html
    - username: kohsamui
    - password: thailand
- Level 33:
    - problem: http://www.pythonchallenge.com/pc/rock/beer.html
    - unlock official solution: http://www.pythonchallenge.com/pcc/rock/gremlins.html
    - username: kohsamui
    - password: thailand