PyCon XIV Gliwice, Poland — programming challenges

Amsterdam Standard

03/11/22

We've prepared few little coding puzzles in **Python** for You! Don't hesitate to tackle them. There might be some gadgets awaiting if You can accomplish them. Good luck.

First task: Put all ducks in a row!

Few fellow ducks have visited us, perhaps You can remember them from old cartoon movies. They arrived out of some order, but we must greet them according to it, to make sure no one is offended.

Switching to the programming world... We've prepared a little code snippet to begin with. We've categorized those ducks into 3 classes - **WeaklingDuck**, **Duck** and **SuperDuck** and created some objects.

Listing below is the entrypoint for You to complete the task.

```
import random
from enum import Enum, auto
from pprint import pp

random.seed(0xDEADBEEF)


class Power(Enum):
    SPEED_BOOTS    = auto()
    STRONG_WILL    = auto()
    MONEY_MAKER    = auto()
    SUPER_STRENGTH = auto()


class WeaklingDuck:
    pass


class Duck:
    def __init__(self, name):
        self.name = name


class SuperDuck(Duck):
    def __init__(self, name, power):
        super().__init__(name)
        if isinstance(power, Power):
            self.power = power
        else:
            self.power = random.choice(list(Power))


def main():
    d1 = WeaklingDuck()
    d2 = Duck("Huey")
    d3 = SuperDuck("Scrooge McDuck", Power.MONEY_MAKER)
    d4 = SuperDuck("Launchpad McQuack", Power.SPEED_BOOTS)
    ducks_in_a_row = [d2, d4, d3, d1]
    pp(ducks_in_a_row)
    # add some code here (and/or edit that above this line)

if __name__ == ’__main__’:
    main()
```

Here’s output so far... not in order and the formatting could be better...

```
[<__main__.Duck object at 0x7f8ddfd0c730>,
 <__main__.SuperDuck object at 0x7f8ddfd0c790>,
 <__main__.SuperDuck object at 0x7f8ddfd0c6d0>,
 <__main__.WeaklingDuck object at 0x7f8ddfe29d30>]
```

Your main goal is to sort them according to these specs (from highest priority to lowest):

SuperDuck with speed boots $\succ$
SuperDuck with any other Power $\succ$
Duck $\succ$
WeaklingDuck

Some rules:

- you can use whatever module from *stdlib*
- you can interfere with existing code - rearrange if needed!
- and don’t forget - also make all the ducks greet themselves nicely!
- bonus: for shortest implementation or clever solution (or both)
- bonus: if you leave as much of existing code untouched, if possible

Second task: Count all the pumpkins!

Halloween is the past tense already for this year, but we need Your help in cleaning out all the mess after the spooky party. We need to find out where are the hidden pumpkins - so we can be sure they will not start to rot and put a quite bad smell to the surroundings.
Again, translating it in the programmer's tongue...
We throw some text at you using generator - one line of fixed length at a time.

```
# some more code here
# ...

def main():
    for line in get_lines():
        # you code goes here

if __name__ == '__main__':
    main()
```

The lines can contain word 'PUMPKIN' (or parts of it). Your task is to count all the PUMPKINs according to these rules (just to make it a little bit more difficult):

Valid count scenarios:

- somewhere in the line

  ```
  fynp[;%)3mPUMPKIN"s~y4nkbx7m^&3b5@[,m^ny
  ```

- same as above, but reversed

  ```
  bwk0\2l:<’n‘dNIKPMUPm=j]5<byzNIKPMUPg5$3
  ```

- span over two lines, where one syllable is at the end of first line and the other one at the beginnig of next line

  ```
  i75s7!yq&=![<?./j$$;1|\dl.>0s’’1|3=2PUMP
  KIN#g65[/^u=a<>>u[{!6{(j8m:,h)<s1hi$@k[9
  ```

Invalid count scenarios:

- at the beginning/end of the line

  ```
  PUMPKINfynp[;%)3m"s~y4nkbx7m^&3b5@[,m^ny
  fynp[;%)3m"s~y4nkbx7m^&3b5@[,m^nyPUMPKIN
  ```

- when some other letter from the word is next to it (left or right)

  ```
  ^o{(zyz"g<*a?UPUMPKINn{8,^=/tote30bkb*#2
  ^o{(zyz"g<*a?PUMPKINKn{8,^=/tote30bkb*#2
  ```

- **extra (*)** in some so called ”trashy” lines, there are all letters from a word PUMPKIN, but in some they are out of order, if that’s the case just skip them:

```
;qqbkrMi+o.=’je!<_m>}_’IvPt"9N^wU[PiK5c|
```

but if they are in order (no matter how many other characters are interleaving them) - that’s a count!

```
;7Pa9%14U:ba$#m-4/x(M:P/bKIf6Nt<,~j1*2’(
```

EXAMPLE - for these lines the count is 2:

```
# c3!hny*,c5pbe3825rsh#m71i7{s^i$*o*g=ePUM
# PKIN/^+3m#(}j@2>8˜nav\f>=PUMPKINs5:PUMPK
# IN*k$on#t.<p]pjia=35‘at2@/PUMPKIN}g3l+*‘
```

The whole input can be found in **sample_input.txt** file. And there is a little helper function that reads it line by line.

Bonus points for:

- making it work if the letters forming a word, would be lowercase
- what if the word changed from ’pumpkin’ to something else? will your code still work?
- try not to use ‘count‘, ‘find‘, ‘rfind‘ functions and ‘in‘ operator

Depending on some implementation details, for the input file You should get 52 or 60 (this is connected with extra case, but both are valid) as a number of pumpkins.

Sample input is at the end of this document.

Third task: First letter / last letter game!

After all the excitement of crazy halloween night and after all fellow ducks have left, we woke up to the harsh reality. To put a little smile back on our faces, we have decided to play some game called *first letter last letter* so don't mind to join us!
The original game runs as follows: one person thinks of a word e.g. *elephan**t***, then another person must find new word that starts with the last letter of the previous word e.g. ***t**rip* and it continues...

There is an input file called **letter_game.txt** that consists of surnames of important people in history of computing. Can you find the longest sequence of those that follow the rules of this game?

We will build a leaderboard - the person who will find the longest sequence wins!

Rules are:

- use Python :)
- sequence must be found programmatically (with some logic) - no hardcoding!
- other than that anything goes!

The solution should be verifiable by the function below - it must return a 2-tuple containing positive number and a True and pass all the assertions:

```
import os
from pathlib import Path

CUR_DIR = os.path.abspath(os.path.dirname(__file__))
INPUT_PATH = Path(CUR_DIR) / ".." / "input" / "letter_game.txt"

def verify(names):
    # 1st check: if all names are unique
    assert len(set(names)) == len(list(names))
    # 2nd check: if all names are in the file
    with open(INPUT_PATH) as f:
        words = set(w.strip().lower() for w in f.readlines() if w.strip())
        assert all(n.lower() in words for n in names)
    # 3rd check: if rules are preserved
    last = None
    c = 0
    for name in names:
        name = name.lower()
        if last is not None and last != name[0]:
            return c, False
        else:
            c += 1
        last = name[-1]
    return c, True

# example:
# print(verify(['Wolfram', 'Matsumoto', 'Odersky']))
# which gives output:
# (3, True)
```

We have some naive algorithm that has found some sequence of length 28 using randomness and recursion. Can You do better?

beck - knuth - hopcroft - tanenbaum - meijer - rossum - meyer - rivest - torvalds - stepanov - von-neumann - ng - gates - stroustrup - patterson - naur - raymond - dean - norvig - graham - musk - kleene - evans - schwartz - zuse - eich - hamming - gosling

Here is the contents of a *letter_game.txt* file for third exercise:

```
lovelace
huffman
wolfram
gosling
codd
steele
felleisen
hoare
naur
zuse
mccarthy
berners-lee
turing
lamport
rossum
thompson
pike
sussman
sethi
karp
evans
karpinski
conway
eich
torvalds
norvig
church
musk
stepanov
ullman
babbage
friedman
floyd
shamir
chomsky
stroustrup
```

```
boole
raymond
fowler
meijer
matsumoto
liskov
kernighan
zuckerberg
beck
strassen
nakamoto
tukey
kay
kruskal
hejlsberg
wirth
stallman
stevens
perlis
knuth
rivest
odersky
sedgewick
shannon
graham
koenig
moore
hickey
warwick
gates
page
curry
minsky
lerdorf
adleman
amdahl
bell
dean
```

```
carmack
ng
hall
dijkstra
lecun
sutherland
leiserson
wall
meyer
peyton-jones
wozniak
hopper
ritchie
hopcroft
von-neumann
iverson
hamming
bellman
booch
schwartz
backus
tanenbaum
romero
kleene
hollerith
petri
aho
patterson
brin
russell
```

Here is an excerpt of *sample_input.txt* for second exercise:

```
,081:c,9g*?}lcp)t7,e0(%|j:dt3\yn-4<!#5o7
q#PU>Mi%6.;Pd""{!0ju0z5u_s’c6*x8#KIjN4_:
yoP^/\_k}g(’&raUdMPk[}.2eKhqI;"2@N\&<6g=
y>fzPmI3<2ujc.K-$qn-y6aU+6)m=PNm&/M"80r@
zf5b92|*php05l9"p_v|k++de6d,,!)(!l%hPUMP
KIN!}‘m>c{qiiPUMPKINe/+i]i*}‘%!zp*<b>NIK
PMUP0&vjp^^hp0a3$~_8n’4:gpw#,2~pr5f2^>6>
g|,r;)a.4v9‘e-/+s~9x==z~rpl1=v13@t4.zsai
yh:<rPU+*M?0wx:‘d<>l(x9{@qjvtvo&Pg+~KIN&
4[t‘02;IU,Kdf-w"x,_ye;q.t$7%(mPN/=P9M5m\
<l6e0yng;PUMPKINv-$&g$b(j")-b4s(|i-.^vf7
gsP~7U5[M\4PjKs‘]I’;i&|2=\{}N[:!|0eg]:@=
\,([>age#p;ao?o&}6k^\&v1!o)~’i[r_vk7‘_-8
#[55u;z$gq/f<s+g)3]xPUMPKIN!0,o-PUMPKIN#
_m=63=(ug1[-k\>}cg{l}PUMPKINik>hval&by@f
m:(l}%]d})8s1p}|ks,[1‘d@,)t\e##"o>a-:|#e
:;kP!l+t5Ur<M]Pz_p:qa?rK\7,!>‘|w7sv‘9I.N
;}mz6;,]qwnc-‘,h;78&8cmi~>PUMPKINzl*3j’!
1.@c"o+"8uxx96a/.jb)vp]f#&=_xf7vve]8ur8r
sz2>NIKPMUP.|r^!!?a&.x:.e+NIKPMUPPUMPKIN
N)@I%($o,P,a?f3M"9K|r&rvm34edPU+*g<~}/cc
ep7#e=m{5lbn,s.84/c<[[_>jx]4\2ui|bf/ukw(
dmky)44aj0u_$y24el0h(h’’*c{gPUMPKIN>,;]>
nw09t|x-g2oh%NIKPMUP(%h=!0’+fc-u.0‘>_%e@
```

For more info about us, please visit our page @ LinkedIn
https://pl.linkedin.com/company/amsterdam-standard
or email us directly: Info @ Amsterdam Standard!