Pydon'ts
Write elegant Python code

by Rodrigo Girão Serrão

EuroPython 2021

About me

Rodrigo Girão Serrão

- Formal education: maths
- Writing Python for 9 years
- Training/teaching:
  - APL (Dyalog Ltd.)
  - Python, maths, etc (mathspp.com)

@mathsppblog

Pydon'ts

Write elegant Python code

Pydon'ts

"Elegance is not a dispensable luxury but a factor that decides between success and failure."

— Edsger Dijkstra,

"Selected Writings on Computing: A Personal Perspective", p.347

Task

```
>>> f("abcdefg")
'aBcDeFg'

>>> f("a cd  g")
'a cD  g'
```

Starting point

```python
def myfunc(a):
    empty=[]
    for i in range(len(a)):
        if i%2==0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Destination

```python
def alternate_casing(text):
    return "".join([
        char.lower() if idx % 2 else char.upper()
        for idx, char in enumerate(text)
    ])
```

Code style matters

Code style matters

```python
def myfunc(a):
    empty=[]
    for i in range(len(a)):
        if i%2==0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Code style matters

```python
def myfunc(a):
    empty=[]
    for i in range(len(a)):
        if i%2==0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Code style matters

```
def myfunc(a):
    empty = []
    for i in range(len(a)):
        if i % 2 == 0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Code style matters

```python
def myfunc(a):
    empty = []
    for i in range(len(a)):
        if i % 2 == 0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Naming matters

Naming matters

```python
def myfunc(a):
    empty = []
    for i in range(len(a)):
        if i % 2 == 0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Naming matters

```
def myfunc(a):
    empty = []
    for i in range(len(a)):
        if i % 2 == 0:
            empty.append(a[i].upper())
        else:
            empty.append(a[i].lower())

    return "".join(empty)
```

Naming matters

```python
def alternate_casing(text):
    letters = []
    for idx in range(len(text)):
        if idx % 2 == 0:
            letters.append(text[idx].upper())
        else:
            letters.append(text[idx].lower())

    return "".join(letters)
```

Naming matters

```python
def alternate_casing(text):
    letters = []
    for idx in range(len(text)):
        if idx % 2 == 0:
            letters.append(text[idx].upper())
        else:
            letters.append(text[idx].lower())

    return "".join(letters)
```

Enumerate me

Enumerate me

Traverse `my_list`:

```
for idx in range(len(my_list)):
    print(my_list[idx])
    ...
```

Enumerate me

Traverse `my_list`:

```python
for elem in my_list:
    print(elem)
    ...
```

Enumerate me

Traverse `my_list` w/ indices:

```
for idx in range(len(my_list)):
    elem = my_list[idx]
    print(idx, elem)
    ...
```

Enumerate me

Traverse `my_list` w/ indices:

```
for idx, elem in enumerate(my_list):
    print(idx, elem)
    ...
```

Enumerate me

```python
def alternate_casing(text):
    letters = []
    for idx in range(len(text)):
        if idx % 2 == 0:
            letters.append(text[idx].upper())
        else:
            letters.append(text[idx].lower())

    return "".join(letters)
```

Enumerate me

```python
def alternate_casing(text):
    letters = []
    for idx in range(len(text)):
        if idx % 2 == 0:
            letters.append(text[idx].upper())
        else:
            letters.append(text[idx].lower())

    return "".join(letters)
```

Enumerate me

```
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            letters.append(char.upper())
        else:
            letters.append(char.lower())

    return "".join(letters)
```

Enumerate me

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            letters.append(char.upper())
        else:
            letters.append(char.lower())

    return "".join(letters)
```

Nest sparingly

Nest sparingly

```
>>> 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
```

Nest sparingly

```
>>> 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.
```

Nest sparingly

```python
try:
    do_some_things()
    this_may_error()
    do_other_things()
except IndexError:
    print("Whoopsie!")
    correct_mistake()
```

Nest sparingly

```
do_some_things()
try:
    this_may_error()
except IndexError:
    print("Whoopsie!")
    correct_mistake()
do_other_things()
```

Nest sparingly

```
with open(filepath, "r") as f:
    contents = f.read()
    process_contents(contents)
```

Nest sparingly

```python
with open(filepath, "r") as f:
    contents = f.read()
process_contents(contents)
```

Nest sparingly

Nesting introduces semantic dependencies.
The less, the better.

Nest sparingly

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            letters.append(char.upper())
        else:
            letters.append(char.lower())

    return "".join(letters)
```

Nest sparingly

```
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            letters.append(char.upper())
        else:
            letters.append(char.lower())

    return "".join(letters)
```

Nest sparingly

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            letters.append(char.upper())
        else:
            letters.append(char.lower())

    return "".join(letters)
```

Nest sparingly

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            capitalised = char.upper()
        else:
            capitalised = char.lower()
        letters.append(capitalised)

    return "".join(letters)
```

Nest sparingly

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            capitalised = char.upper()
        else:
            capitalised = char.lower()
        letters.append(capitalised)

    return "".join(letters)
```

Truthy, Falsy, and bool

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if len(my_list) > 0:
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if 3 > 0:
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if True:
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if len(my_list) > 0:
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if my_list:
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if bool(my_list):
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
>>> my_list = [1, 3, 73]
>>> if True:
...     print(my_list[-1])
... else:
...     print("No last element ;(")
...
73
```

Truthy, Falsy, and bool

```
0
0.0
None
""
[]
{}
tuple()
set()
```

Truthy, Falsy, and bool

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            capitalised = char.upper()
        else:
            capitalised = char.lower()
        letters.append(capitalised)

    return "".join(letters)
```

Truthy, Falsy, and bool

```
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            capitalised = char.upper()
        else:
            capitalised = char.lower()
        letters.append(capitalised)

    return "".join(letters)
```

Truthy, Falsy, and bool

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2 == 0:
            capitalised = char.upper()
        else:
            capitalised = char.lower()
        letters.append(capitalised)

    return "".join(letters)
```

Truthy, Falsy, and bool

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2:
            capitalised = char.lower()
        else:
            capitalised = char.upper()
        letters.append(capitalised)

    return "".join(letters)
```

Truthy, Falsy, and bool

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2:
            capitalised = char.lower()
        else:
            capitalised = char.upper()
        letters.append(capitalised)

    return "".join(letters)
```

Conditional expressions

Conditional expressions

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        if idx % 2:
            capitalised = char.lower()
        else:
            capitalised = char.upper()
        letters.append(capitalised)

    return "".join(letters)
```

Conditional expressions

```python
if idx % 2:
    capitalised = char.lower()
else:
    capitalised = char.upper()
letters.append(capitalised)
```

Conditional expressions

```
capitalised = char.lower() if idx % 2 else char.upper()
letters.append(capitalised)
```

Conditional expressions

```
letters.append(
    char.lower() if idx % 2 else char.upper()
)
```

Conditional expressions

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        letters.append(
            char.lower() if idx % 2 else char.upper()
        )

    return "".join(letters)
```

List comprehensions

List comprehensions

```python
my_list = []
for elem in iter:
    my_list.append(func(elem))
```

List comprehensions

```
my_list = [func(elem) for elem in iter]
```

List comprehensions

```python
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        letters.append(
            char.lower() if idx % 2 else char.upper()
        )

    return "".join(letters)
```

List comprehensions

```
def alternate_casing(text):
    letters = []
    for idx, char in enumerate(text):
        letters.append(
            char.lower() if idx % 2 else char.upper()
        )

    return "".join(letters)
```

List comprehensions

```
def alternate_casing(text):
    letters = [
        char.lower() if idx % 2 else char.upper()
        for idx, char in enumerate(text)
    ]

    return "".join(letters)
```

List comprehensions

```python
def alternate_casing(text):
    letters = [
        char.lower() if idx % 2 else char.upper()
        for idx, char in enumerate(text)
    ]

    return "".join(letters)
```

List comprehensions

```python
def alternate_casing(text):
    return "".join([
        char.lower() if idx % 2 else char.upper()
        for idx, char in enumerate(text)
    ])
```

References

- Pydon'ts:
  - Bite-sized refactoring, https://mathspp.com/blog/pydonts/bite-sized-refactoring
  - Does elegance matter, https://mathspp.com/blog/pydonts/does-elegance-matter
  - Code style matters, https://mathspp.com/blog/pydonts/code-style-matters
  - Enumerate me, https://mathspp.com/blog/pydonts/enumerate-me
  - Truthy, Falsy, and bool, https://mathspp.com/blog/pydonts/truthy-falsy-and-bool

gum.co/pydonts

@mathsppblog

mathspp.com/subscribe

/mathspp/talks

$$\overbrace{\underbrace{\text{rodrigo}}_{\text{name}}\text{@}\underbrace{\text{mathspp.com}}_{\text{site}}}^{\text{email}}$$