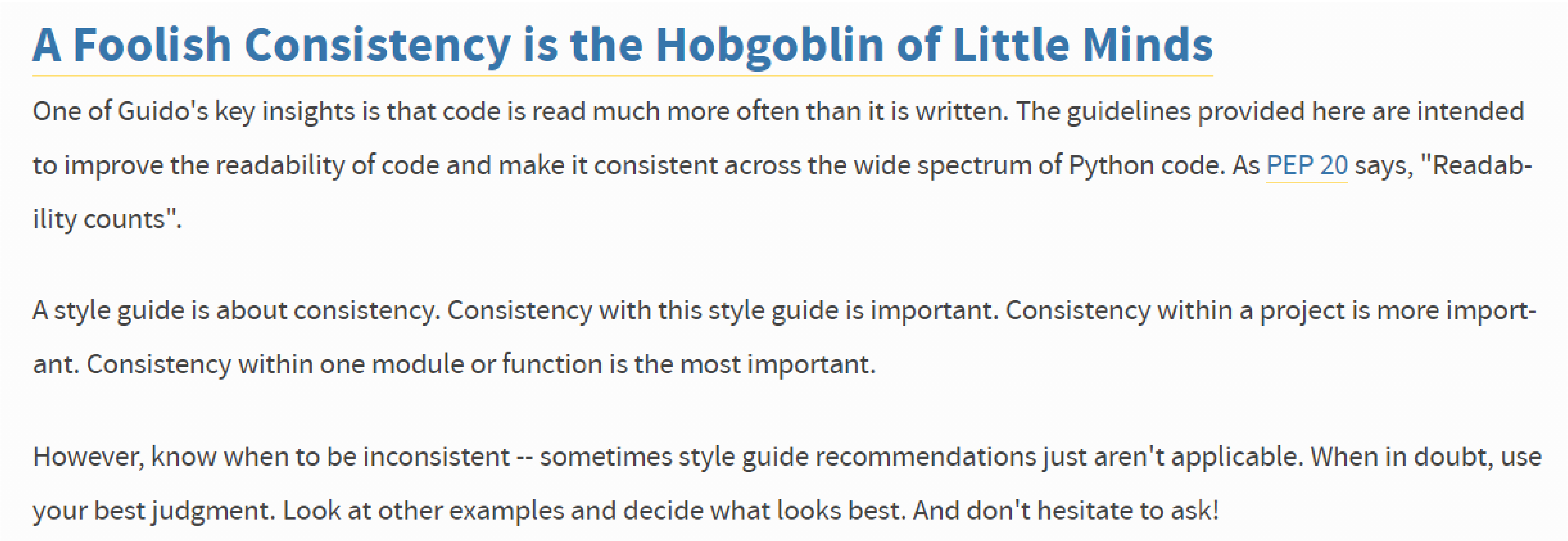
Lecture 5: Functions

Copy and paste is the enemy!

The PEP8 Style Guide

https://www.python.org/dev/peps/pep-0008/



A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".

A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

However, know when to be inconsistent -- sometimes style guide recommendations just aren't applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask!

Functions

What is a function?

A function is a block of code that does not run unless it is **called**, which can be done as many times as desired.

Functions

What is a function?

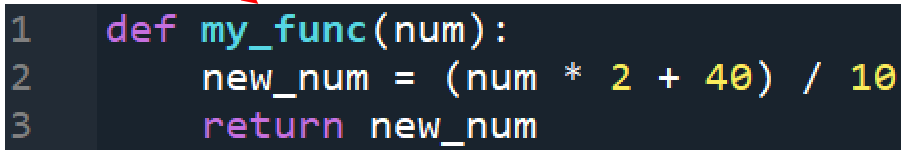
A function is a block of code that does not run unless it is **called**, which can be done as many times as desired.

Why use a function?

- Repeated code (avoiding copy-paste)
- Organizing code

Function Basics

Name in lower case with underscores

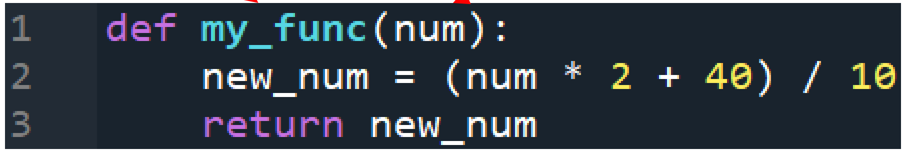


```python
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```

Function Basics

Name in lower case with underscores

0-n positional arguments

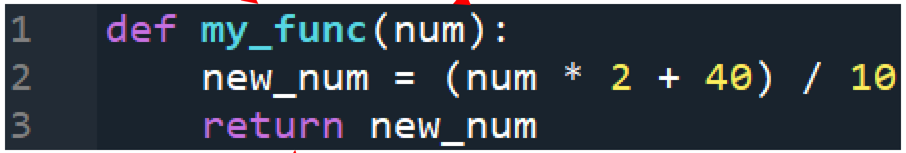


```
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```

Function Basics

Name in lower case with underscores

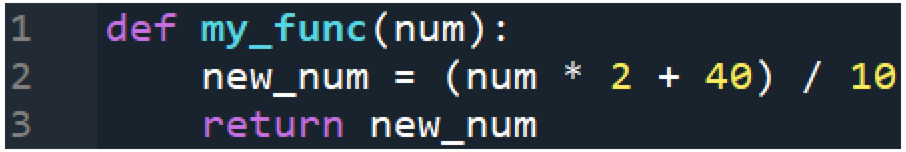
0-n positional arguments



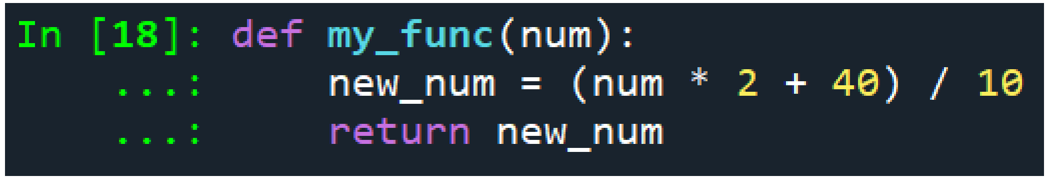
```python
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```

Return statement – defaults to returning *None* dtype if not included

Function Basics

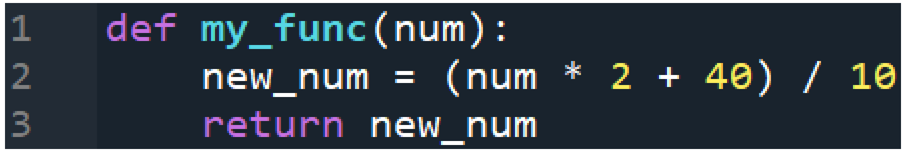


```python
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```

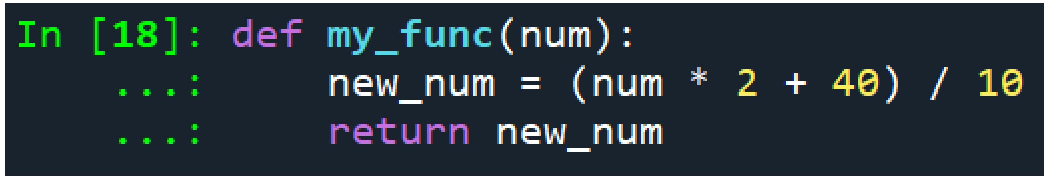


```
In [18]: def my_func(num):
    ...:     new_num = (num * 2 + 40) / 10
    ...:     return new_num
```

Function Basics



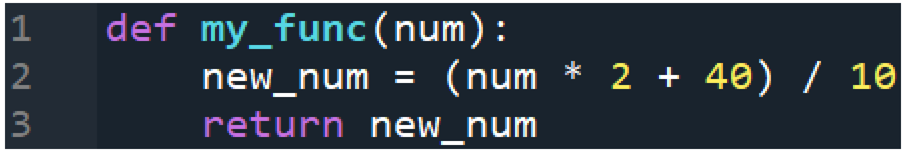
```python
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```



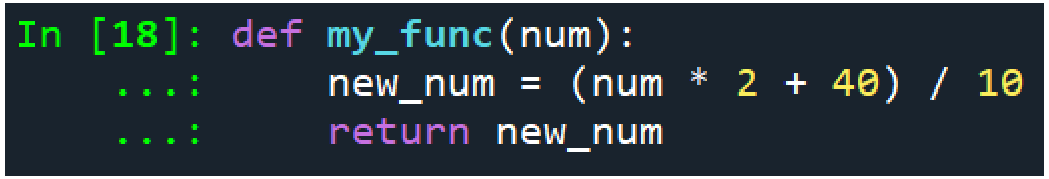
```
In [18]: def my_func(num):
    ...:     new_num = (num * 2 + 40) / 10
    ...:     return new_num
```

No code is executed during a function declaration!

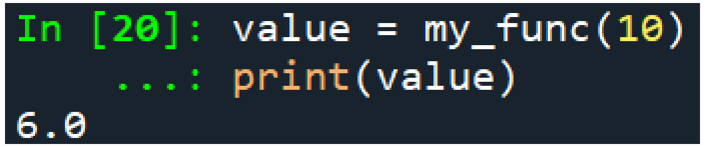
Function Basics



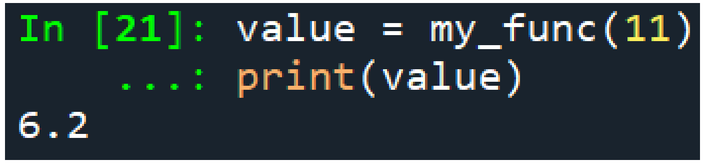
```
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```



```
In [18]: def my_func(num):
    ...:     new_num = (num * 2 + 40) / 10
    ...:     return new_num
```

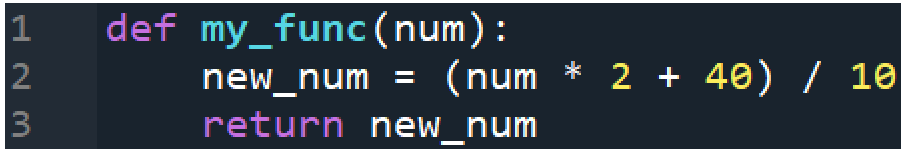


```
In [20]: value = my_func(10)
    ...: print(value)
6.0
```

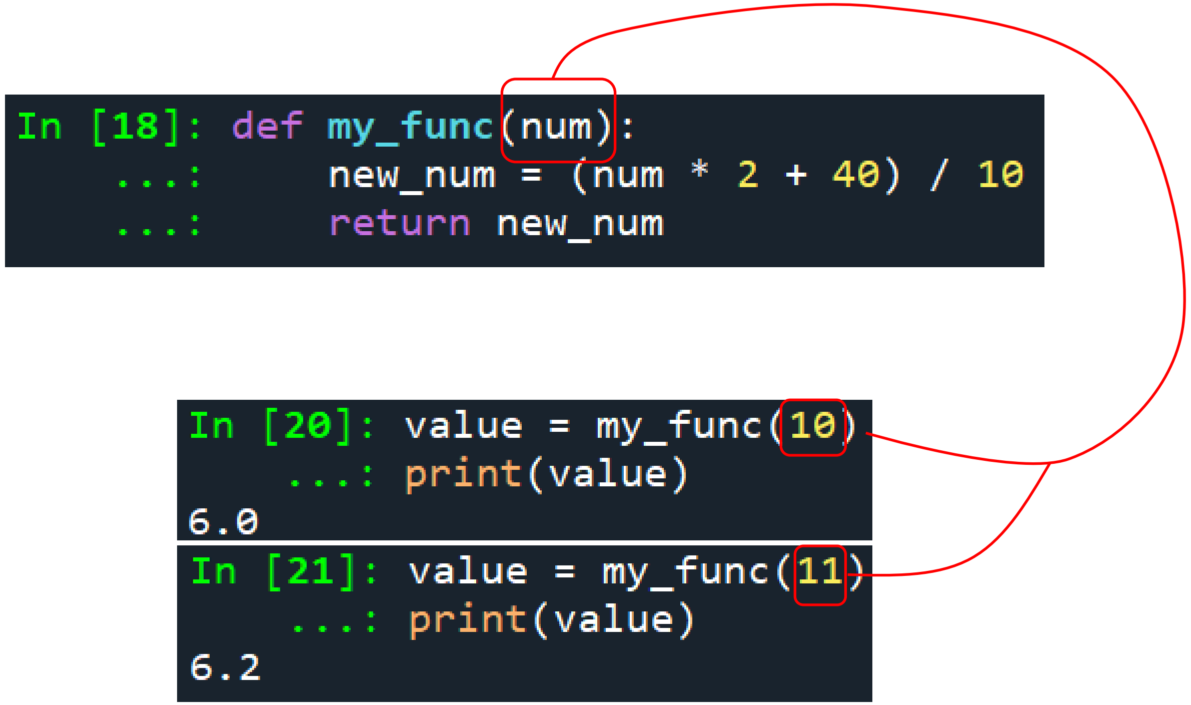


```
In [21]: value = my_func(11)
    ...: print(value)
6.2
```

Function Basics



```
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```

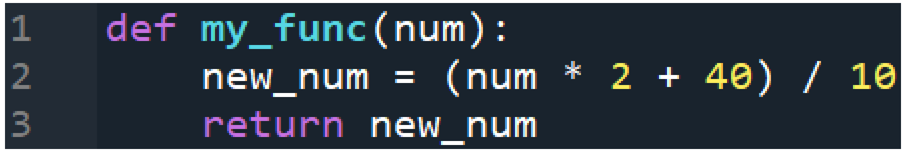


```
In [18]: def my_func(num):
    ...:     new_num = (num * 2 + 40) / 10
    ...:     return new_num
```

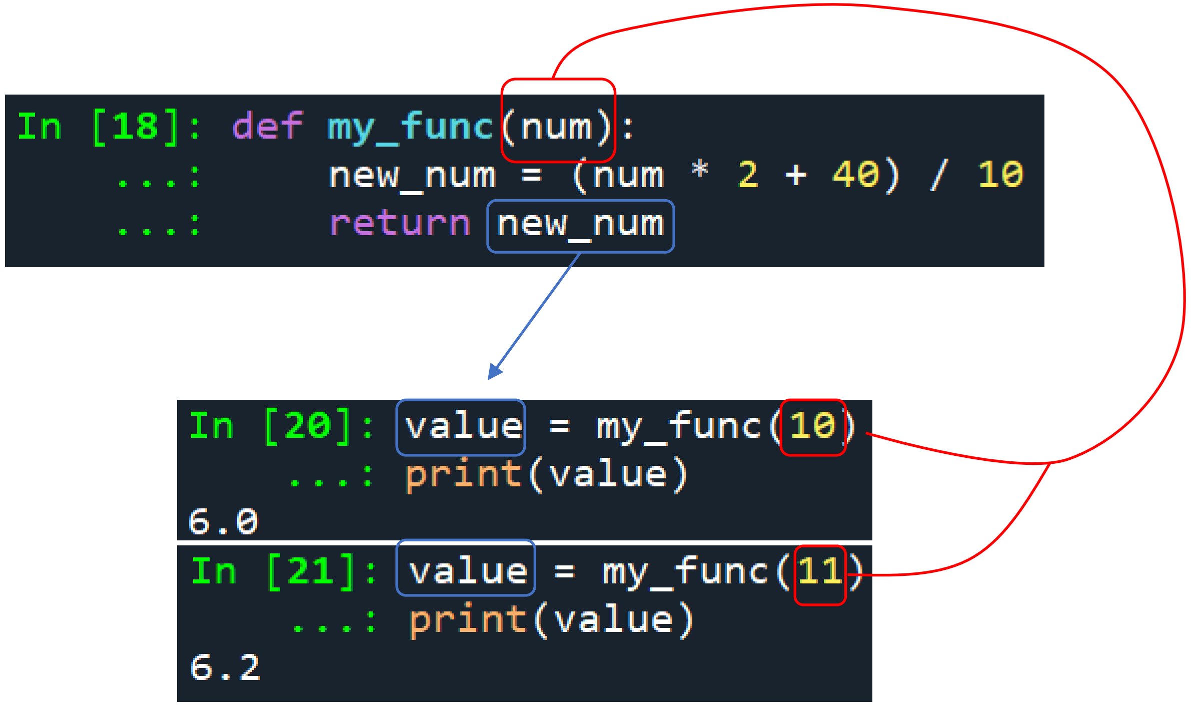
```
In [20]: value = my_func(10)
    ...: print(value)
6.0
```

```
In [21]: value = my_func(11)
    ...: print(value)
6.2
```

Function Basics



```python
def my_func(num):
    new_num = (num * 2 + 40) / 10
    return new_num
```

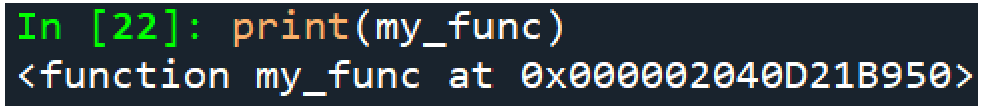


```
In [18]: def my_func(num):
    ...:     new_num = (num * 2 + 40) / 10
    ...:     return new_num
```

```
In [20]: value = my_func(10)
    ...: print(value)
6.0
In [21]: value = my_func(11)
    ...: print(value)
6.2
```

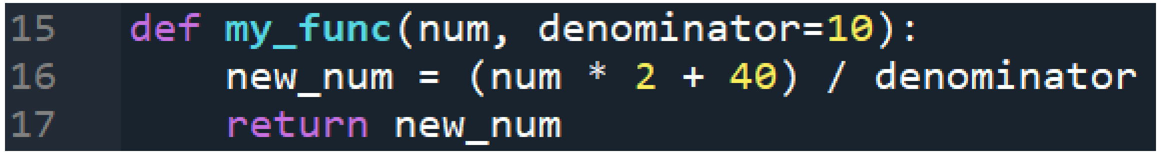
Function Basics

Functions themselves are objects, just like strings or integers or anything else. If you do not call it by ending in (), you will only see the object itself.

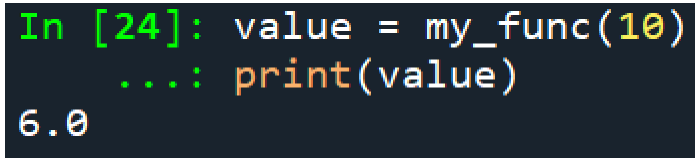


```
In [22]: print(my_func)
<function my_func at 0x000002040D21B950>
```

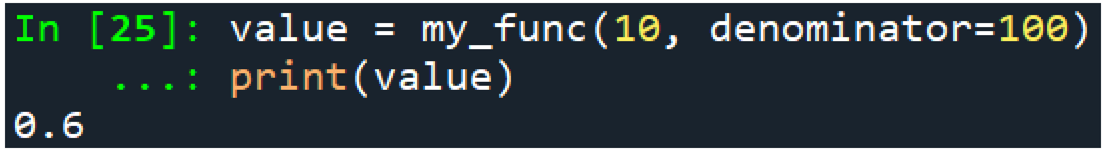
Functions and key word arguments



```python
def my_func(num, denominator=10):
    new_num = (num * 2 + 40) / denominator
    return new_num
```



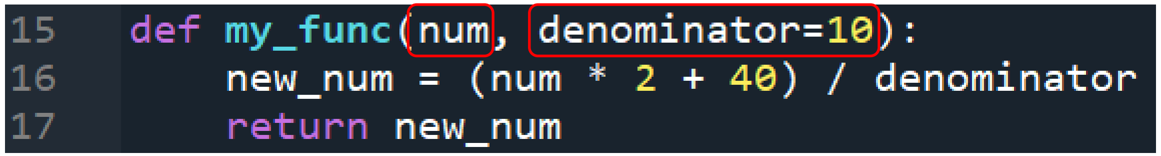
```
In [24]: value = my_func(10)
    ...: print(value)
6.0
```



```
In [25]: value = my_func(10, denominator=100)
    ...: print(value)
0.6
```

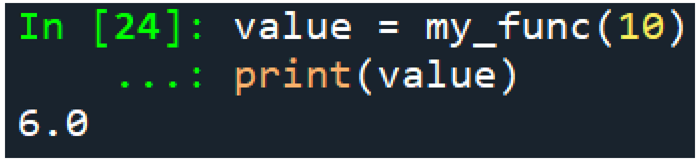
Functions and key word arguments

arg kwarg



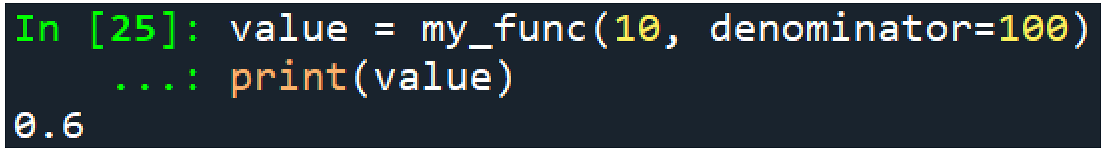
```
def my_func(num, denominator=10):
    new_num = (num * 2 + 40) / denominator
    return new_num
```

arg



```
In [24]: value = my_func(10)
    ...: print(value)
6.0
```

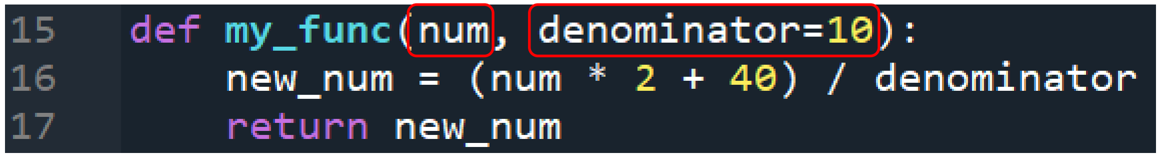
arg kwarg



```
In [25]: value = my_func(10, denominator=100)
    ...: print(value)
0.6
```

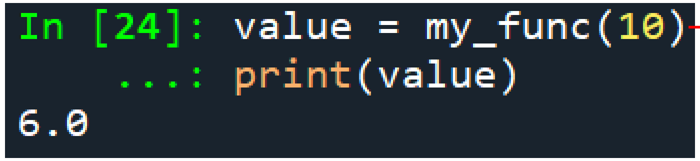
Functions and key word arguments

arg kwarg



```python
def my_func(num, denominator=10):
    new_num = (num * 2 + 40) / denominator
    return new_num
```

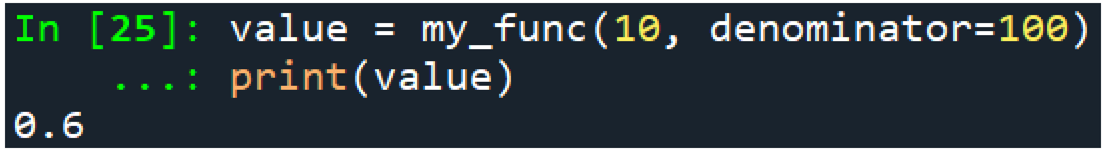
arg



```
In [24]: value = my_func(10)
    ...: print(value)
6.0
```

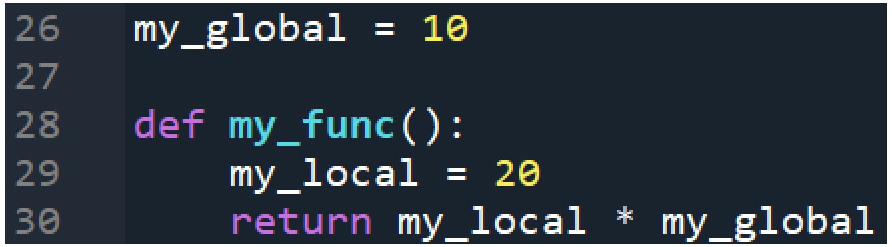
Equivalent to writing:
my_func(10, denominator=10)

arg kwarg

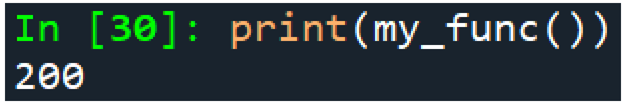


```
In [25]: value = my_func(10, denominator=100)
    ...: print(value)
0.6
```

Functions and name-space



```python
my_global = 10

def my_func():
    my_local = 20
    return my_local * my_global
```

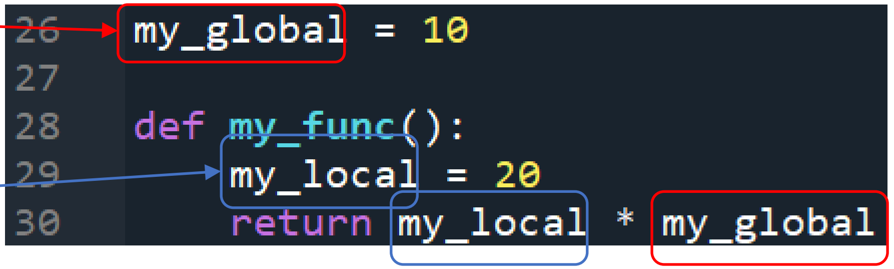


```
In [30]: print(my_func())
200
```

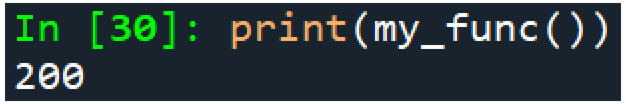
Functions and name-space

Global variable

Local variable

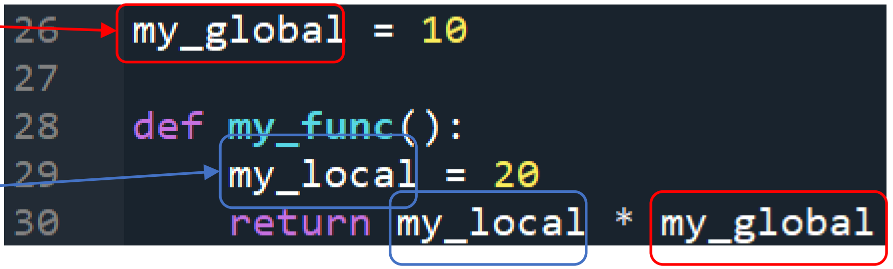


```python
my_global = 10

def my_func():
    my_local = 20
    return my_local * my_global
```



```
In [30]: print(my_func())
200
```

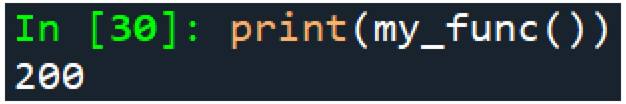
Functions and name-space



Global variable

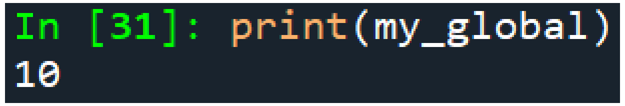
Local variable

```python
my_global = 10

def my_func():
    my_local = 20
    return my_local * my_global
```



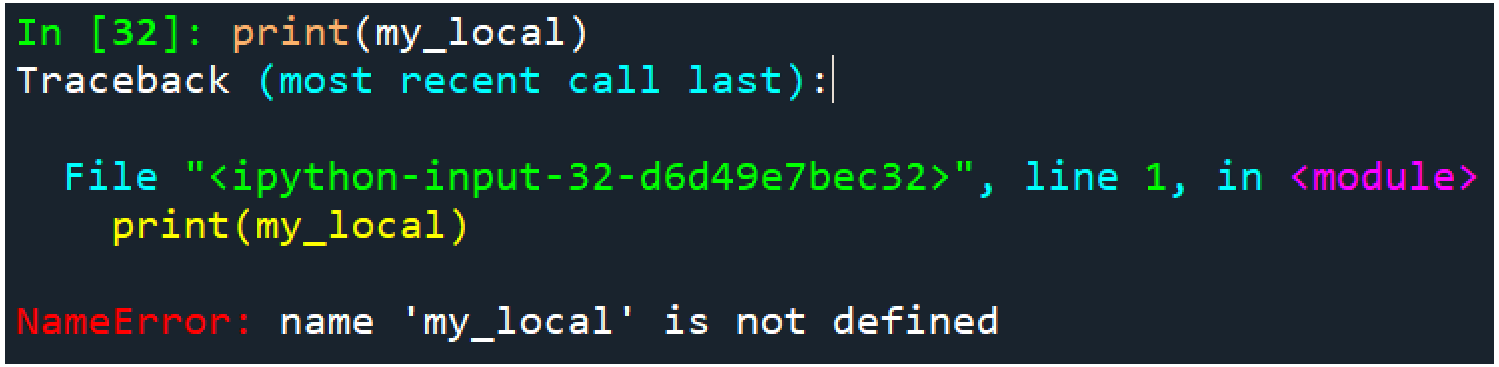
```
In [30]: print(my_func())
200
```

Global variable



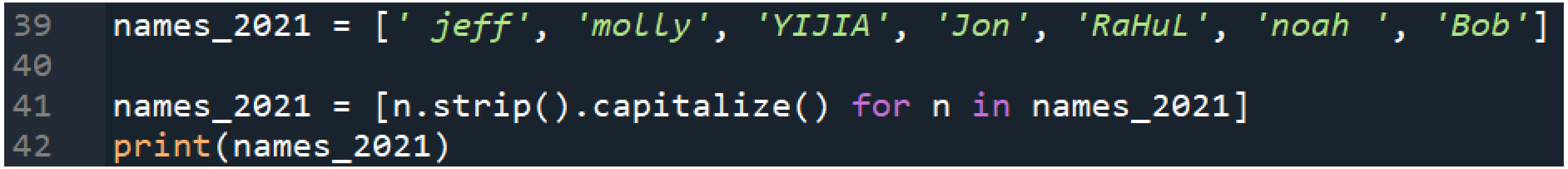
```
In [31]: print(my_global)
10
```

Local variable

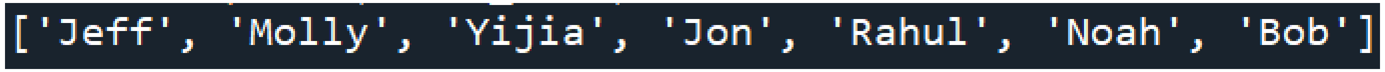


```
In [32]: print(my_local)
Traceback (most recent call last):

  File "<ipython-input-32-d6d49e7bec32>", line 1, in <module>
    print(my_local)

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

Functions: an example

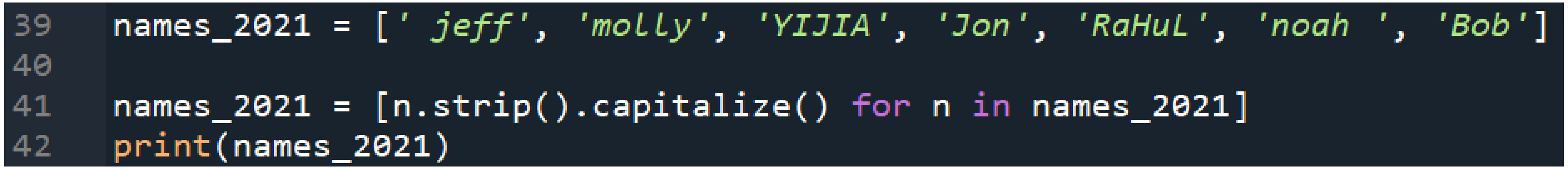


```python
names_2021 = [' jeff', 'molly', 'YIJIA', 'Jon', 'RaHuL', 'noah ', 'Bob']

names_2021 = [n.strip().capitalize() for n in names_2021]
print(names_2021)
```

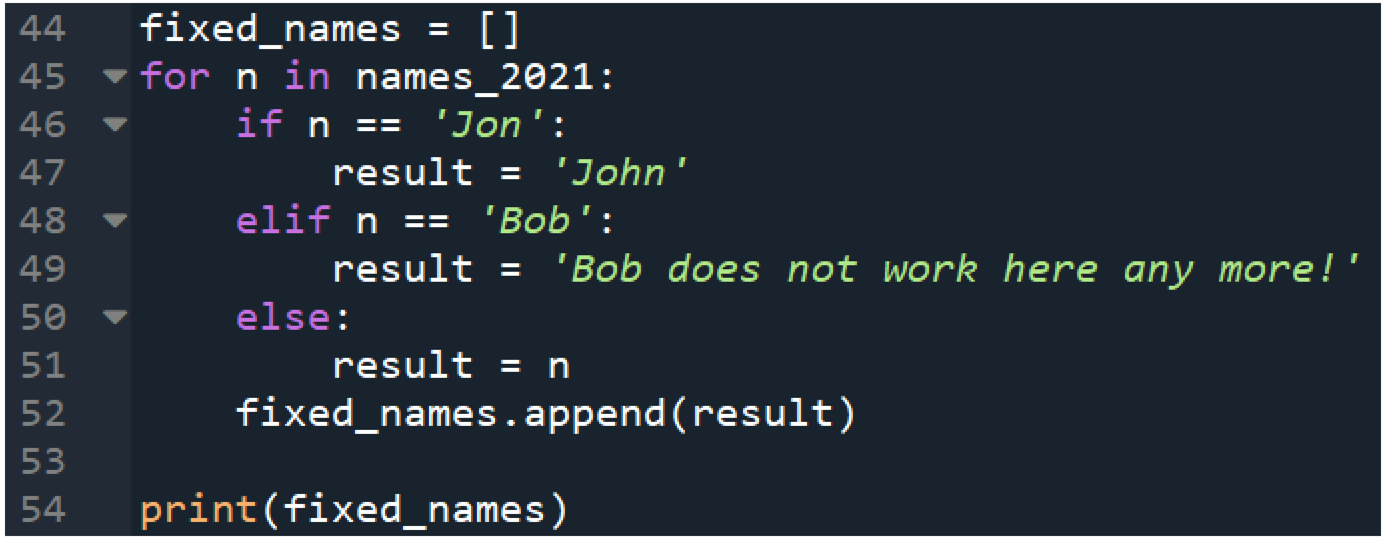


```
['Jeff', 'Molly', 'Yijia', 'Jon', 'Rahul', 'Noah', 'Bob']
```

Functions: an example

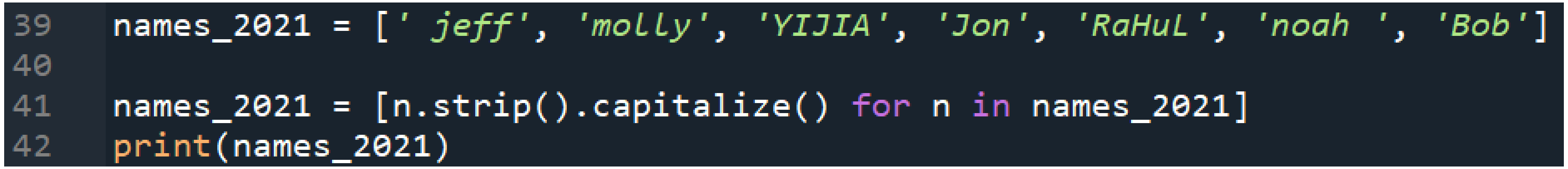


```python
names_2021 = [' jeff', 'molly', 'YIJIA', 'Jon', 'RaHuL', 'noah ', 'Bob']

names_2021 = [n.strip().capitalize() for n in names_2021]
print(names_2021)
```

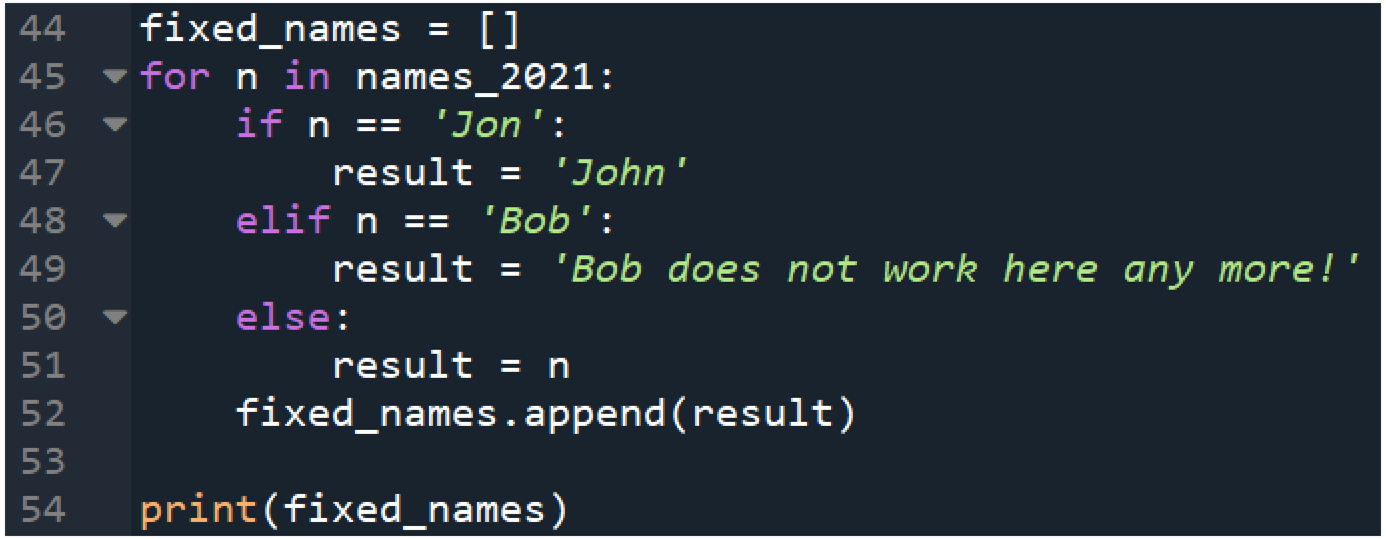


```python
fixed_names = []
for n in names_2021:
    if n == 'Jon':
        result = 'John'
    elif n == 'Bob':
        result = 'Bob does not work here any more!'
    else:
        result = n
    fixed_names.append(result)

print(fixed_names)
```

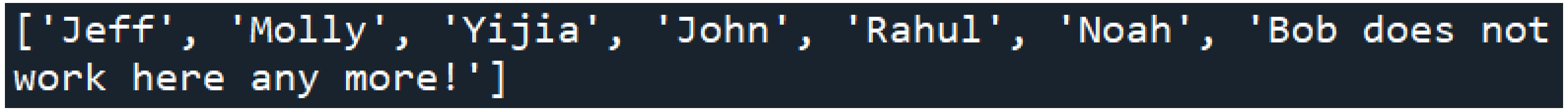
Functions: an example



```python
names_2021 = [' jeff', 'molly', 'YIJIA', 'Jon', 'RaHuL', 'noah ', 'Bob']

names_2021 = [n.strip().capitalize() for n in names_2021]
print(names_2021)
```

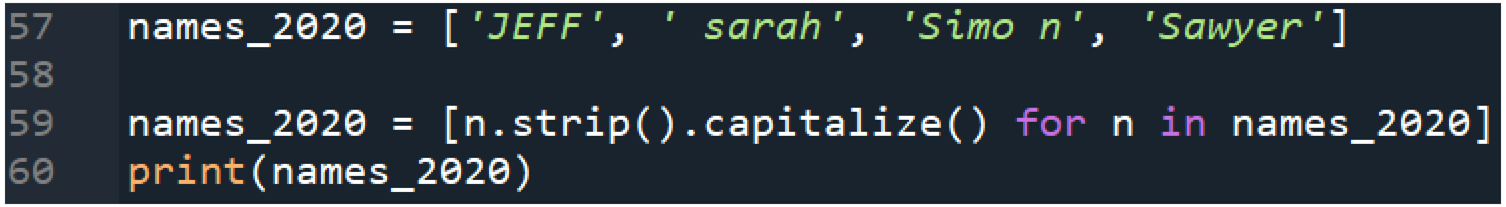


```python
fixed_names = []
for n in names_2021:
    if n == 'Jon':
        result = 'John'
    elif n == 'Bob':
        result = 'Bob does not work here any more!'
    else:
        result = n
    fixed_names.append(result)

print(fixed_names)
```

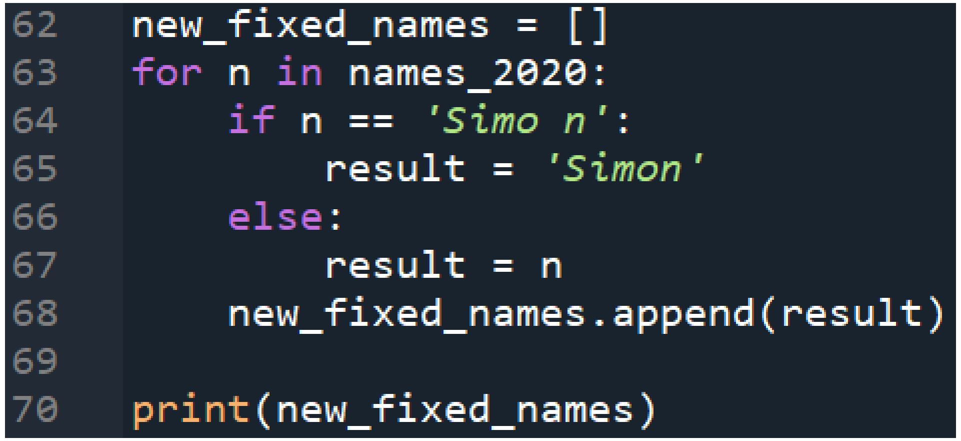


```
['Jeff', 'Molly', 'Yijia', 'John', 'Rahul', 'Noah', 'Bob does not
work here any more!']
```

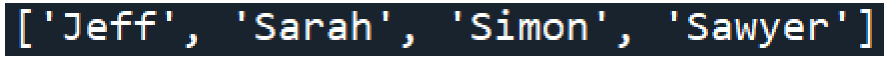
Functions: an example



```python
names_2020 = ['JEFF', ' sarah', 'Simo n', 'Sawyer']

names_2020 = [n.strip().capitalize() for n in names_2020]
print(names_2020)
```



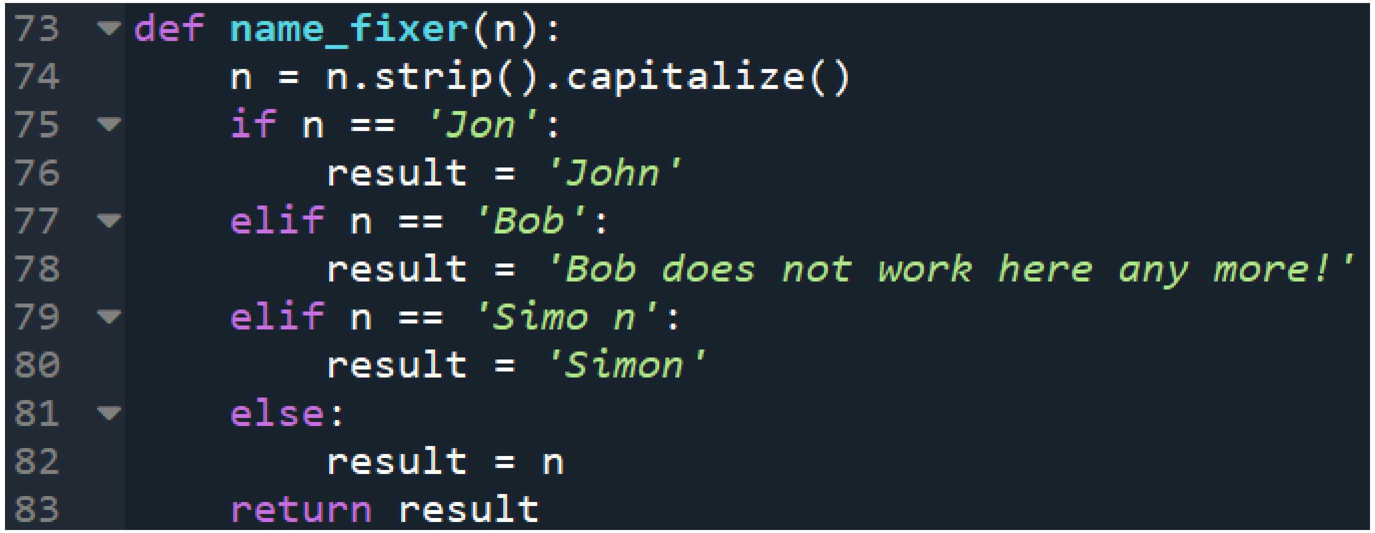
```python
new_fixed_names = []
for n in names_2020:
    if n == 'Simo n':
        result = 'Simon'
    else:
        result = n
    new_fixed_names.append(result)

print(new_fixed_names)
```



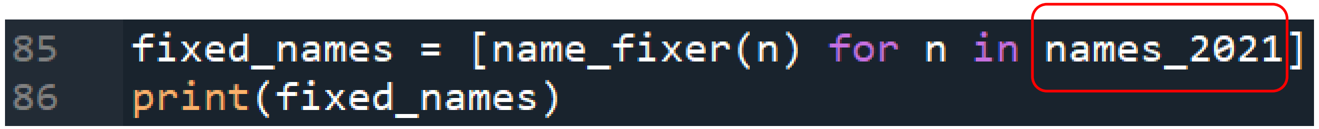
```
['Jeff', 'Sarah', 'Simon', 'Sawyer']
```

Functions: an example

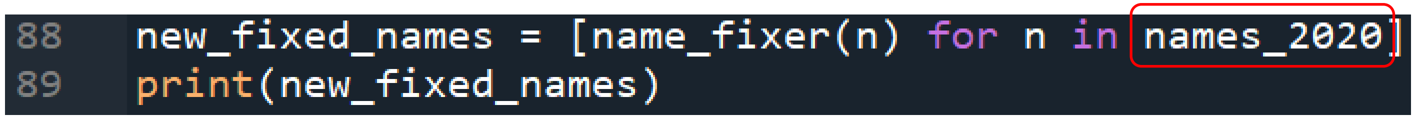
Turning all those steps into one function:



```python
def name_fixer(n):
    n = n.strip().capitalize()
    if n == 'Jon':
        result = 'John'
    elif n == 'Bob':
        result = 'Bob does not work here any more!'
    elif n == 'Simo n':
        result = 'Simon'
    else:
        result = n
    return result
```



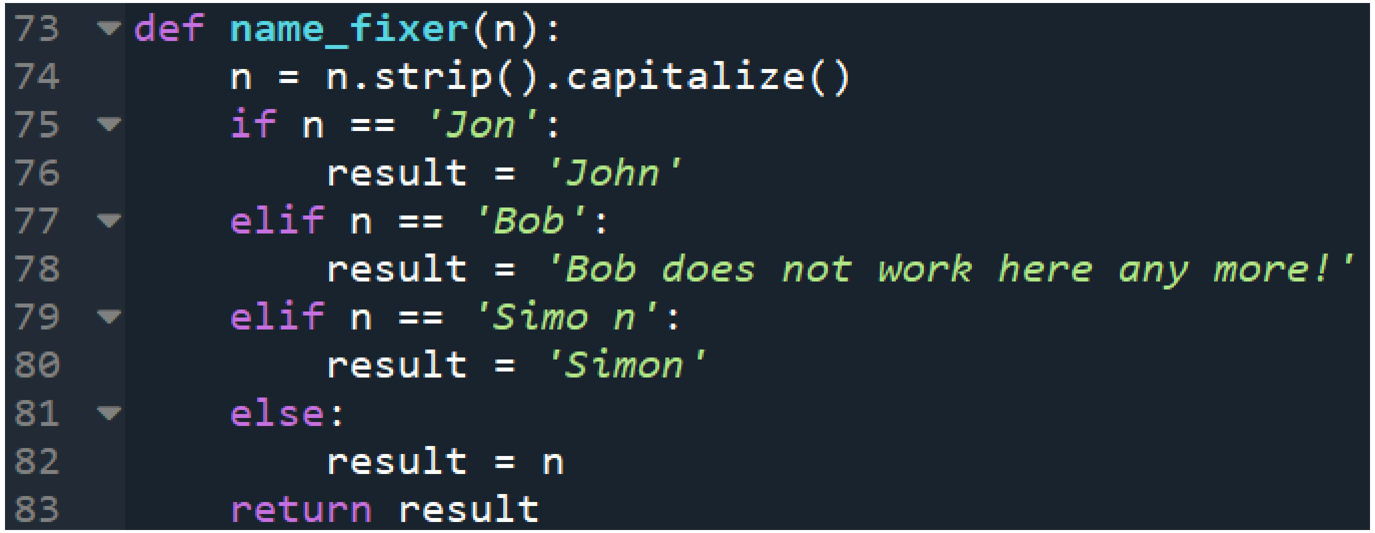
```python
fixed_names = [name_fixer(n) for n in names_2021]
print(fixed_names)
```



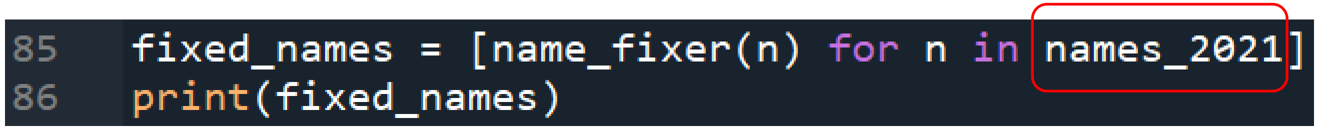
```python
new_fixed_names = [name_fixer(n) for n in names_2020]
print(new_fixed_names)
```

Functions: an example

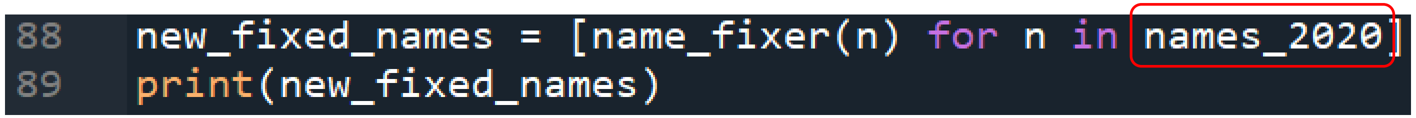
Turning all those steps into one function:



```python
def name_fixer(n):
    n = n.strip().capitalize()
    if n == 'Jon':
        result = 'John'
    elif n == 'Bob':
        result = 'Bob does not work here any more!'
    elif n == 'Simo n':
        result = 'Simon'
    else:
        result = n
    return result
```



```python
fixed_names = [name_fixer(n) for n in names_2021]
print(fixed_names)
```



```python
new_fixed_names = [name_fixer(n) for n in names_2020]
print(new_fixed_names)
```

Is “result” defined globally?

Is “n” defined globally?

Is “fixed_names” defined inside the function?

Functions overview

```
my_global = 100
def my_func(a, b=0):
    answer = a * b
    return answer
```

- Global variable
- Function name
- Argument (arg)
- Key-word argument (kwarg)
- Local variable
- Value when function is called

- Variables can be accessed upward, but not downward, e.g. globals can be seen inside a function, but locals cannot be seen at the global level
- Can be 0-N of both args and kwargs, but all args must come before any kwargs
- If no return statement, function has implicit “`return None`”