PYTHON DATA SCIENCE TOOLBOX I

Scope and user-defined functions

Crash course on scope in functions

- Not all objects are accessible everywhere in a script
- Scope - part of the program where an object or *name* may be accessible
  - Global scope - defined in the main body of a script
  - Local scope - defined inside a function
  - Built-in scope - names in the pre-defined built-ins module

Global vs. local scope (1)

```
In [1]: def square(value):
   ...:     """Returns the square of a number."""
   ...:     new_val = value ** 2
   ...:     return new_val

In [2]: square(3)
Out[2]: 9

In [3]: new_val
---------------------------------------------------------------------
NameError                           Traceback (most recent call last)
<ipython-input-3-3cc6c6de5c5c> in <module>()
----> 1 new_value
NameError: name 'new_val' is not defined
```

Global vs. local scope (2)

```
In [1]: new_val = 10

In [2]: def square(value):
   ...:     """Returns the square of a number."""
   ...:     new_val = value ** 2
   ...:     return new_val

In [3]: square(3)
Out[3]: 9

In [4]: new_val
Out[4]: 10
```

Global vs. local scope (3)

```
In [1]: new_val = 10

In [2]: def square(value):
   ...:     """Returns the square of a number."""
   ...:     new_value2 = new_val ** 2
   ...:     return new_value2

In [3]: square(3)
Out[3]: 100

In [4]: new_val = 20

In [5]: square(3)
Out[5]: 400
```

Global vs. local scope (4)

```
In [1]: new_val = 10

In [2]: def square(value):
   ...:     """Returns the square of a number."""
   ...:     global new_val
   ...:     new_val = new_val ** 2
   ...:     return new_val

In [3]: square(3)
Out[3]: 100

In [4]: new_val
Out[4]: 100
```

PYTHON DATA SCIENCE TOOLBOX I

Let's practice!

PYTHON DATA SCIENCE TOOLBOX I

Nested functions

Nested functions (1)

nested.py

```
def outer( … ):
    """ … """
    x = …

    def inner( … ):
        """ … """
        y = x ** 2

    return …
```

Nested functions (2)

square3.py

```
def raise_both(value1, value2):
    """Raise value1 to the power of value2
    and vice versa."""

    new_value1 = value1 ** value2
    new_value2 = value2 ** value1

    new_tuple = (new_value1, new_value2)

    return new_tuple
```

Nested functions (3)

mod2plus5.py

```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values."""

    def inner(x):
        """Returns the remainder plus 5 of a value."""
        return x % 2 + 5

    return (inner(x1), inner(x2), inner(x3))
```

```
In [1]: print(mod2plus5(1, 2, 3))
(6, 5, 6)
```

Returning functions

raise.py

```
def raise_val(n):
    """Return the inner function."""

    def inner(x):
    """Raise x to the power of n."""
        raised = x ** n
        return raised

    return inner
```

```
In [1]: square = raise_val(2)
In [2]: cube = raise_val(3)
In [3]: print(square(2), cube(4))
4 64
```

Using nonlocal

nonlocal.py

```
def outer():
    """Prints the value of n."""
    n = 1

    def inner():
        nonlocal n
        n = 2
        print(n)

    inner()
    print(n)
```

```
In [1]: outer()
2
2
```

Scopes searched

- Local scope
- Enclosing functions
- Global
- Built-in

PYTHON DATA SCIENCE TOOLBOX I

Let's practice!

PYTHON DATA SCIENCE TOOLBOX I

Default and flexible arguments

You'll learn:

- Writing functions with default parameters
- Using flexible arguments
  - Pass any number of arguments to a functions

Add a default argument

```
In [1]: def power(number, pow=1):
  ....:     """Raise number to the power of pow."""
  ....:     new_value = number ** pow
  ....:     return new_value

In [2]: power(9, 2)
Out[2]: 81

In [3]: power(9, 1)
Out[3]: 9

In [4]: power(9)
Out[4]: 9
```

Flexible arguments: *args (1)

add_all.py

```
def add_all(*args):
    """Sum all values in *args together."""

    # Initialize sum
    sum_all = 0

    # Accumulate the sum
    for num in args:
        sum_all += num

    return sum_all
```

Flexible arguments: *args (2)

```
In [1]: add_all(1)
Out[1]: 1

In [2]: add_all(1, 2)
Out[2]: 3

In [3]: add_all(5, 10, 15, 20)
Out[3]: 50
```

Flexible arguments: **kwargs

```
In [1]: print_all(name="Hugo Bowne-Anderson", employer="DataCamp")
name: Hugo Bowne-Anderson
employer: DataCamp
```

Flexible arguments: **kwargs

kwargs.py

```
def print_all(**kwargs):
    """Print out key-value pairs in **kwargs."""

    # Print out the key-value pairs
    for key, value in kwargs.items():
        print(key + ": " + value)
```

```
In [1]: print_all(name="dumbledore", job="headmaster")
job: headmaster
name: dumbledore
```

PYTHON DATA SCIENCE TOOLBOX I

Let's practice!

PYTHON DATA SCIENCE TOOLBOX I

Bringing it all together

Next exercises:

- Generalized functions:
  - Count occurrences for any column
  - Count occurrences for an arbitrary number of columns

Add a default argument

power.py

```
def power(number, pow=1):
    """Raise number to the power of pow."""
    new_value = number ** pow
    return new_value
```

```
In [1]: power(9, 2)
Out[1]: 81

In [2]: power(9)
Out[2]: 9
```

Flexible arguments: *args (1)

add_all.py

```
def add_all(*args):
    """Sum all values in *args together."""

    # Initialize sum
    sum_all = 0

    # Accumulate the sum
    for num in args:
        sum_all = sum_all + num

    return sum_all
```

PYTHON DATA SCIENCE TOOLBOX I

Let's practice!