Warm Up

20 minutes

1. **Each person in the group review 3 random classes or functions in a favorite portfolio project.**

2. **Each person answer the following**:
   a. Is your class/func understandable by ‘yourself’?
   b. Would your code be understandable by ‘others’?
   c. Would a potential employer find your coding style **‘consistent’**?

3. **Share your findings with your groupmates and then with the class in the SPD 2.3 slack channel.**

Refactoring - PEP 8

ACS 4931

Learning Outcomes

By the end of today, you should be able to…

1. Understand the importance of code readability and style guides.
2. Implement Python code according to the PEP8 style guide.
3. Refactor code according to PEP8 to improve code readability.

Why Use a Style Guide?

- "Code is read much more often than it is written" - Guido van Rossum
- An attempt to improve readability
- All about consistency
- Everyone uses the PEP 8 Style Guide in Python

Code Layout

- Save Python script files using UTF-8 encoding
- Indentation
    - Spaces are preferred indentation method.
    - Use 4 spaces per indentation level.
- Maximum Line Length → 79 characters
    - easier for working on two files side-by-side
    - easier for code review/comparison
- Continuation lines
    - See the next slide for visual explanation!

Indentation (Continuation lines)

How would you improve the readability of the following code?

```
# Reference: From PEP 8
# Indentation

# Wrong:


def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
    return 23.20

var_one = 3
var_two = 1
var_three = -9
var_four = 3
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

These are continuation lines.

Indentation (Continuation lines) - Wrong

```
# Reference: From PEP 8
# Indentation

# Wrong:

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
    return 23.20


# Arguments on first line forbidden when not using vertical alignment.
var_one = 3
var_two = 1
var_three = -9
var_four = 3
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

Indentation (Continuation lines) - Refactored

```python
# Reference: From PEP 8
# Indentation
# Correct:

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
    return 23.20

# Aligned with opening delimiter.
var_one = 3
var_two = 1
var_three = -9
var_four = 3
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# or
# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

4 spaces added to the argument list to distinguish the function body.

OR

Indentation

7 mins

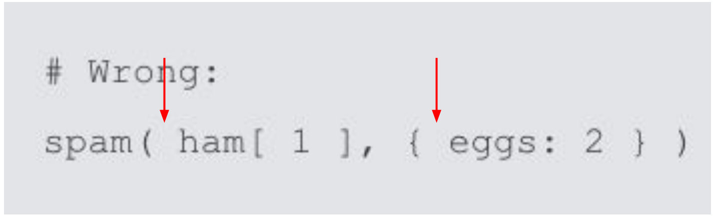
Click here to view the starter code.

1. Clone the repo and navigate to the pep8-refactor activity.
2. Solve **Activity 1: Indentation.**

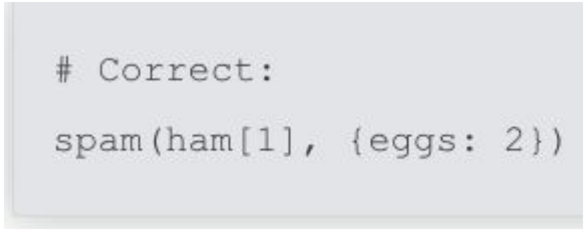
Break (10 mins)

Whitespaces

- Blank Lines
  - Surround top-level function and class definitions with two blank lines.
  - Method definitions inside a class are surrounded by a single blank line
- Avoid whitespaces immediately inside parentheses, brackets or braces



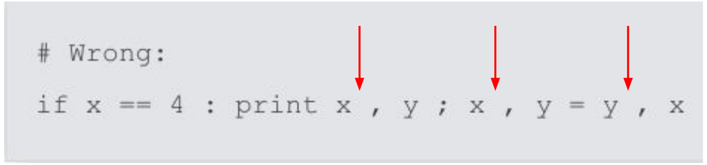
```
# Wrong:
spam( ham[ 1 ], { eggs: 2 } )
```



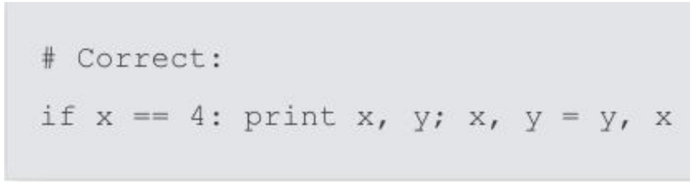
```
# Correct:
spam(ham[1], {eggs: 2})
```

Whitespaces

- Avoid whitespaces immediately before a comma, semicolon, or colon

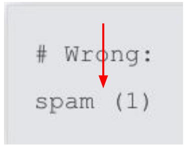


```
# Wrong:
if x == 4 : print x , y ; x , y = y , x
```

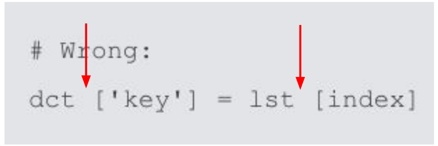


```
# Correct:
if x == 4: print x, y; x, y = y, x
```

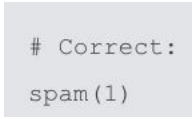
Avoid whitespaces immediately before the open parenthesis that starts the argument list of a function call or, starts an indexing or slicing



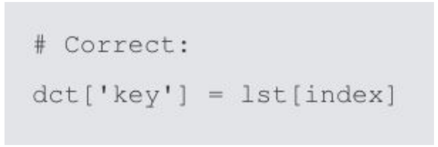
```
# Wrong:
spam (1)
```



```
# Wrong:
dct ['key'] = lst [index]
```



```
# Correct:
spam(1)
```



```
# Correct:
dct['key'] = lst[index]
```

Naming Conventions

- Names to avoid:
    - Never use the characters 'l' (lowercase letter L), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
- **Class** names should normally use the CapWords convention.
    - E.g. “PlayerBehaviour”, “CharacterController”.
- **Variable** names and **function** names should be lowercase, with words separated by underscores as necessary to improve readability.
    - E.g. “my_health”, “read_file()”, “find_closest_enemy()”, “take_damage()”, “speed_limit”.
- Constants are written in all capital letters with underscores separating words.
    - E.g. “MAX_OVERFLOW” and “TOTAL”.

Naming Conventions

- Public attributes should have no leading underscores.
- Always use ***self*** for the first argument to **instance** methods.
- Always use ***cls*** for the first argument to **class** methods (@classmethod).

Whitespaces and Naming Convention

```
"""
This is a guess game. Guess what number the computer has chosen!
"""
import random
lowerLimit=0
UPPER_limit= 100
randomNumber = random.randint (lowerLimit ,UPPER_limit )
print ('The computer has selected a number between 0 and 100. Use your supernatural superpowers to guess what the number is.')
while True:
   UserGuess = int (input("Enter a number between 0 and 100 (including): "))
   if UserGuess> randomNumber:
       print ("Guess a smaller number.")
   elif UserGuess <randomNumber:
       print ("Guess a larger number.")
   else: #UserGuess == randomNumber:
       print ("You Won!")
       break
```

Whitespaces, Naming Convention (Refactored)

```
"""
This is a guess game. Guess what number the computer has chosen!
"""
import random


LOWER_LIMIT = 0 # assuming these are supposed to stay the same no matter what.
UPPER_LIMIT = 100
random_number = random.randint(LOWER_LIMIT, UPPER_LIMIT)
print('The computer has selected a number between 0 and 100. Use your '
      'supernatural superpowers to guess what the number is.')


while True:
    user_guess = int(input("Enter a number between 0 and 100 (including): "))

    if user_guess > random_number:
        print("Guess a smaller number.")
    elif user_guess < random_number:
        print("Guess a larger number.")
    else: #user_guess == random_number:
        print("You Won!")
```

Whitespaces and Naming convention

12 min

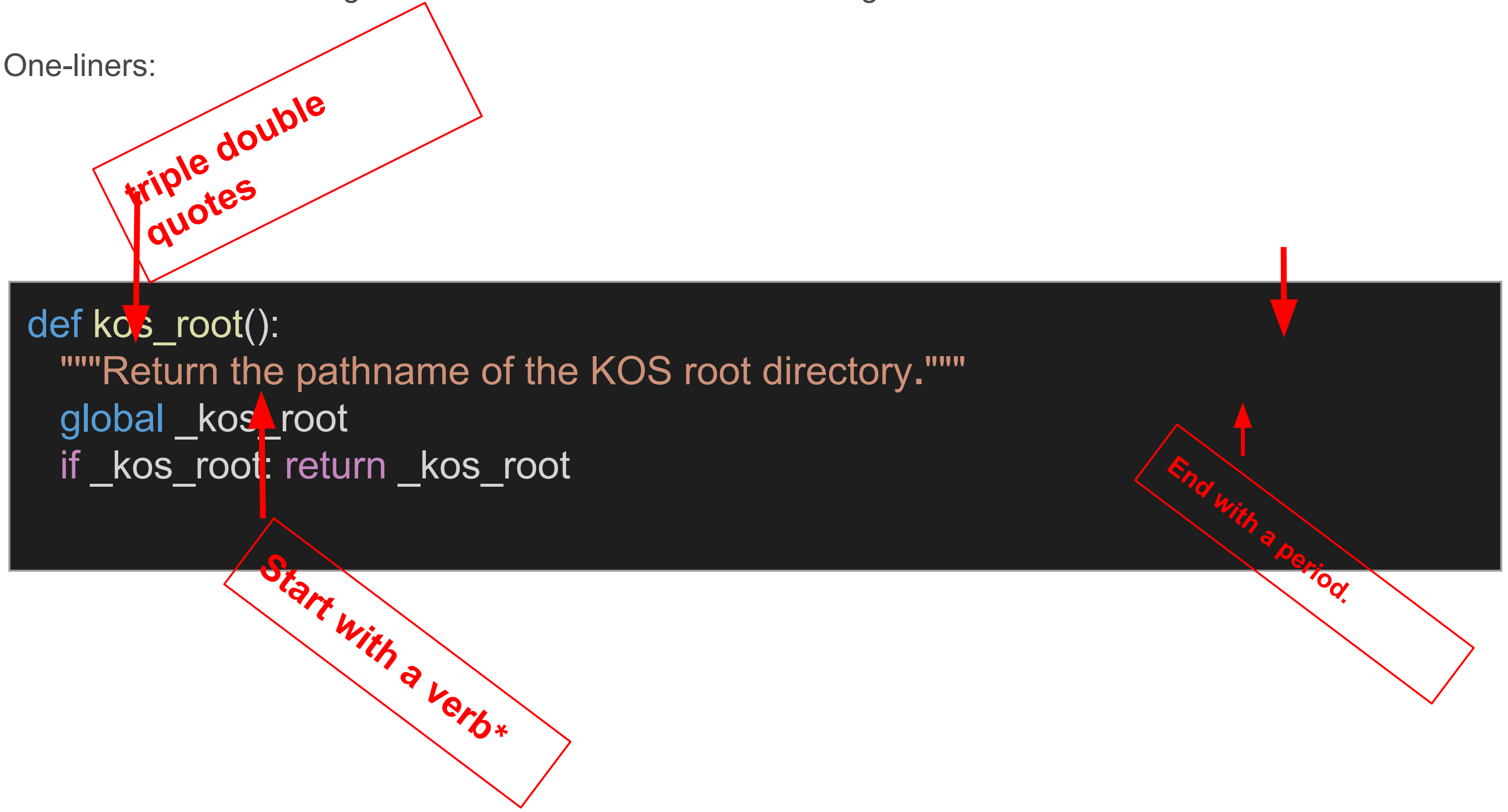
Click here for the activity.

1. Solve **Activity 2: Whitespaces and Naming Convention**

Docstrings

There are two forms of docstrings: 1. One-liners 2. Multi-line docstrings

1. One-liners:



```python
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    if _kos_root: return _kos_root
```

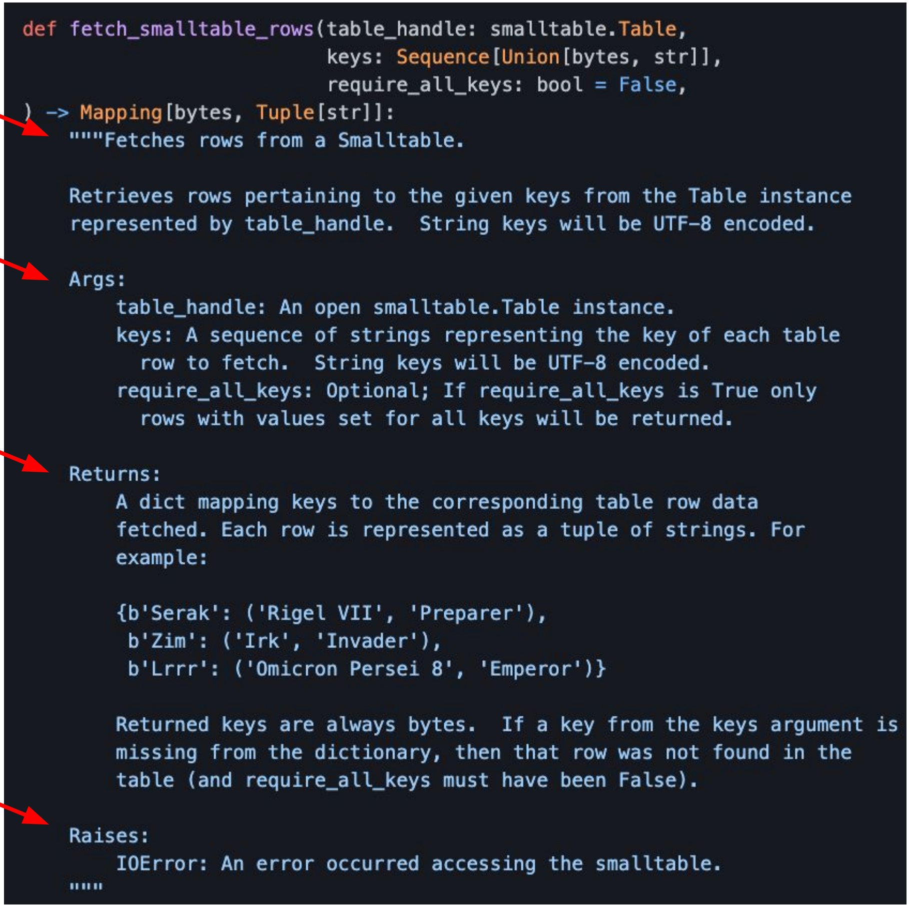
Docstrings

2. Multi-line docstrings:

```
def foo():
  """A multi-line
  docstring.
  """


def bar():
  """
  A multi-line
  docstring.
  """


def complex(real=0.0, imag=0.0):
  """Form a complex number.

  Keyword arguments:
  real -- the real part (default 0.0)
  imag -- the imaginary part (default 0.0)
  """
  if imag == 0.0 and real == 0.0:
      return complex_zero
  ...
```

Docstrings

MAKE SCHOOL



```
def fetch_smalltable_rows(table_handle: smalltable.Table,
                          keys: Sequence[Union[bytes, str]],
                          require_all_keys: bool = False,
) -> Mapping[bytes, Tuple[str]]:
    """Fetches rows from a Smalltable.

    Retrieves rows pertaining to the given keys from the Table instance
    represented by table_handle.  String keys will be UTF-8 encoded.

    Args:
        table_handle: An open smalltable.Table instance.
        keys: A sequence of strings representing the key of each table
          row to fetch.  String keys will be UTF-8 encoded.
        require_all_keys: Optional; If require_all_keys is True only
          rows with values set for all keys will be returned.

    Returns:
        A dict mapping keys to the corresponding table row data
        fetched. Each row is represented as a tuple of strings. For
        example:

        {b'Serak': ('Rigel VII', 'Preparer'),
         b'Zim': ('Irk', 'Invader'),
         b'Lrrr': ('Omicron Persei 8', 'Emperor')}

        Returned keys are always bytes.  If a key from the keys argument is
        missing from the dictionary, then that row was not found in the
        table (and require_all_keys must have been False).

    Raises:
        IOError: An error occurred accessing the smalltable.
    """
```

Google Python Style Guide

Module Docstrings

```python
"""This script prompts a user to enter a message to encode or decode
using a classic Caesar shift substitution (3 letter shift)."""

import string

shift = 3
choice = raw_input("would you like to encode or decode?")
word = (raw_input("Please enter text"))
letters = string.ascii_letters + string.punctuation + string.digits
encoded = ''
if choice == "encode":
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
        else:
            x = letters.index(letter) + shift
            encoded = encoded + letters[x]
if choice == "decode":
    for letter in word:
        if letter == ' ':
            encoded = encoded + ' '
        else:
            x = letters.index(letter) - shift
            encoded = encoded + letters[x]

print encoded
```

Docstrings

You notice a function does not have a docstring. Add a docstrings.

```
"""
Reference: https://math.ryerson.ca/~danziger/professor/MTH141/Handouts/projections.pdf
"""


import numpy as np
def proj_u_onto_v(u,v):
   return np.dot(u, v)/np.dot(v, v) * v



force = np.array([1,1])
displacement = np.array([1,0])
proj = proj_u_onto_v(force, displacement)
print(proj[0], proj[1])
```

Docstrings - Refactored

```
"""
Reference: https://math.ryerson.ca/~danziger/professor/MTH141/Handouts/projections.pdf
"""


import numpy as np
def proj_u_onto_v(u,v):
   """Returns the projection of u vector onto v vector."""
   return np.dot(u, v)/np.dot(v, v) * v

force = np.array([1,1])
displacement = np.array([1,0])
proj = proj_u_onto_v(force, displacement)
print(proj[0], proj[1])
```

Docstrings for class

```
class fighter:
  def __init__(self):
     self.x = 0
     self.y = 0
     self.health = 100
     self.gas = 3
  def set_position(self, x, y):
     self.x = x
     self.y = y
  def render(self):
     print("rendered the fighter")
  def take_damage(self, damage):
     self.health -= damage
     if self.health <= 0:
        print("the figher fell apart!")
  def consume_gas(self):
     if self.gas > 0:
        self.gas -= 1
     else:
        print('out of gas! ')
  def take_gas(self, gas_amound):
     self.gas += gas_amound

f = fighter()
f.set_position(10, 10)
f.consume_gas()
f.take_damage(90)
f.take_damage(40)
```

Docstrings for class - Refactored

```
"""
PEP 8: Surround top-level function and class definitions with two blank
lines. Method definitions inside a class are surrounded
by a single blank line.
"""

class Fighter:
  """Represents a Fighter airplane in a 2D top-down game."""
  def __init__(self):
    self.x = 0
    self.y = 0
    self.health = 100
    self.gas = 3 # current level of fuel

  def set_position(self, x, y):
    self.x = x
    self.y = y

  def render(self):
    """Renders the entire fighter."""
    print("rendered the fighter")

  def take_damage(self, damage):
    """Reduces health by 'damage' amount."""
    self.health -= damage
    if self.health <= 0:
      print("the fighter fell apart!")

  def consume_gas(self):
    """Reduces gas level by 1 unit."""
    if self.gas > 0:
      self.gas -= 1
    else:
      print('out of gas! ')

  def take_gas(self, gas_amound):
    """Increases gas level by 'gas_amount'."""
    self.gas += gas_amound


f = Fighter()
f.set_position(10, 10)
f.consume_gas()
f.take_damage(90)
f.take_damage(40)
```

Docstrings

10 mins

Click here for the activity.

1. Solve **Activity 3: Docstrings.**

Summary

- Consistency is the most important style!
- PEP 8 provides a guideline so it become easier for developers to understand each other’s code.
- Proper Indentation, whitespaces, naming convention and docstrings can significantly increase code readability.
- Sometimes, there are deviations from PEP8 based on the company you work for. **Be sure to listen to your manager and ultimately follow conventions already present in the codebase**!

References and Further Study

1. PEP 8 Style Guide for Python Code.
2. Google Python Style Guide
3. Pylint Documentation