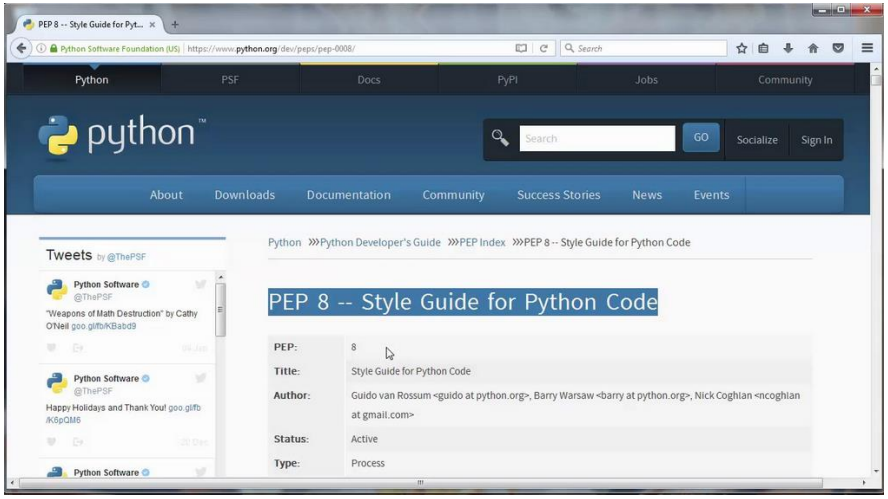
Computer Programming in Python

Python Programming Standards

The PEP 8 Style Guide

Python Programming Standards

- Python Coding Style (PEP 8)
  - Idiomatic Python code is often referred to as being *Pythonic*
  - Python Enhancement Proposal 8 (PEP 8) is the style guide for Python Code.
    - Authors: Guido van Rossum, Barry Warsaw, Nick Coghlan



Python Programming Standards

- Python Style Guide PEP 8
  - The PEP 8 Style Guide for Python is an evolving, comprehensive list of recommended rules and styles for developing Python programs
  - Those that are important and commonly used for Python 3.x are covered here

Python Programming Standards

- Python Style Guide PEP 8
  - A style guide is about consistency
  - Consistency in style is important
  - Consistency within a project is more important
  - Consistency within one module or function is the most important

Python Programming Standards

- Code Lay-out
  - Indentation
    - Although PEP 8 recommends using 4 spaces per indentation level:
      - Most IDEs are ***tab*** oriented
      - Python 3 and above does not allow spaces and tabs to be mixed
    - Since Python is indentation specific, best to use the tab key consistently

Python Programming Standards

- Code Lay-out
  - Import statements
    - Should be at the top of the file
    - Should be on separate lines
    - Recommended order
      - Standard Library imports
      - Related third party imports
      - Local application imports
    - Wildcard imports are discouraged

```
# Correct:
import os
import sys
```

```
# Wrong:
import sys, os
```

Python Programming Standards

- String Quotes
  - In Python, single-quoted strings and double-quoted strings are the same
  - PEP 8 does not make a recommendation for this
    - Pick a rule and stick to it
  - However, when a string contains single or double quote characters, use the other one to avoid backslashes in the string
    - This improves readability
  - For triple-quoted strings, always use double quote characters

Python Programming Standards

- White Space
  - Avoid extraneous white space

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

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

```
# Correct:
x = 1
y = 2
long_variable = 3
```

```
# Wrong:
x             = 1
y             = 2
long_variable = 3
```

  - Always surround binary operators with a single space on either side

Python Programming Standards

- Compound Statements
  - On the same line are discouraged

```
# Wrong:
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

```
# Correct:
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

  - This includes conditional statements and loops

```
# Wrong:
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

Python Programming Standards

- Comments
  - Comments that contradict the code are worse than no comments
  - Update comments when the code changes!
  - Comments should be complete sentences
  - The first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers!)
  - Block comments generally consist of one or more paragraphs built out of complete sentences, with each sentence ending in a period

```
# Comment using complete sentences.
```

Python Programming Standards

- Naming Conventions
  - Package and module (file) names
    - Modules should have short, all-lowercase names
    - Underscores can be used in the module name if it improves readability
    - Python packages should also have short, all-lowercase names, although the use of underscores is discouraged
  - Class Names
    - Class names should normally use the CapWords convention (uppercasing with first letter uppercase)

Python Programming Standards

- Naming Conventions
  - Variable names
    - Should be lowercase, with words separated by underscores as necessary to improve readability.
  - Function names
    - Should be lowercase, with words separated by underscores as necessary to improve readability.
  - Method names
    - Should be lowercase, with words separated by underscores as necessary to improve readability.

Note that these are all the same

Python Programming Standards

- Naming Conventions
  - Constants
    - Use all capital letters with underscores separating words

    Examples: MAX_OVERFLOW and TOTAL.

  - Exception Names
    - Should use the uppercasing convention also known as the CapWords convention
    - Since exceptions should be classes, they should follow the convention for classes

Python Programming Standards

- Naming Conventions
  - Function and Method Arguments
    - Always use *self* for the first argument to instance methods
    - Always use *cls* for the first argument to class methods

Python Programming Standards

- Function Arguments
  - Can be passed to functions in four different ways
    1. Positional arguments (mandatory)
    2. Keyword arguments
    3. Arbitrary argument list
    4. Arbitrary keyword argument dictionary

Python Programming Standards

- **Positional arguments** are mandatory and have no default values
  - They are the simplest form of arguments and they can be used for the few function arguments that are fully part of the function's meaning and their order is natural
  - For instance, in send(message, recipient) or point(x, y) the user of the function has no difficulty remembering that those two functions require two arguments, and in which order

send(message, recipient)

Python Programming Standards

- **Positional arguments** are mandatory and have no default values
  - In the two previous cases, it is possible to use argument names when calling the functions
  - It is possible to switch the order of arguments, calling for instance send(recipient='World', message='Hello') and point(y=2, x=1) but this reduces readability compared to the more straightforward calls:

    send('Hello', 'World') and point(1, 2).

Python Programming Standards

- **Keyword arguments** are not mandatory and have default values
  - They are often used for optional parameters sent to the function
  - When a function has more than two or three positional parameters, its signature is more difficult to remember and using keyword arguments with default values is helpful
    - For instance, a more complete send function could be defined as send(message, to, cc=None, bcc=None)
    - Here cc and bcc are optional, and evaluate to None when they are not passed another value

Python Programming Standards

- **Keyword arguments** are not mandatory and have default values
  - Calling a function with keyword arguments can be done in multiple ways in Python
    - It is possible to follow the order of arguments in the definition without explicitly naming the arguments, like in send('Hello', 'World', 'Cthulhu', ‘You'), sending a blind carbon copy to You
    - It would also be possible to name arguments in another order, like in send('Hello again', 'World', bcc=‘You', cc='Cthulhu')
    - Those two possibilities are better avoided. Follow the syntax that is the closest to the function definition: send('Hello', 'World', cc='Cthulhu', bcc=‘You')

Python Programming Standards

- The **arbitrary argument list** is the third way to pass arguments to a function
  - If the function intention is better expressed by a signature with an extensible number of positional arguments, it can be defined with the *args constructs
  - In the function body, args will be a tuple of all the remaining positional arguments
  - For example, send(message, *args) can be called with each recipient as an argument: send('Hello', ‘You', 'Mom', 'Cthulhu'), and in the function body args will be equal to (‘You', 'Mom', 'Cthulhu')
- However, this construct has some drawbacks and should be used with caution

Python Programming Standards

- The **arbitrary keyword argument dictionary** is the last way to pass arguments to functions
  - If the function requires an undetermined series of named arguments, it is possible to use the **kwargs construct
  - In the function body, kwargs will be a dictionary of all the passed named arguments that have not been caught by other keyword arguments in the function signature
- The same caution as in the case of *arbitrary argument list* is necessary, for similar reasons: these powerful techniques are to be used when there is a proven necessity to use them, and they should not be used if the simpler and clearer construct is sufficient to express the function's intention

Python Programming Standards

- Function Arguments
  - It is up to the programmer writing the function to determine which arguments are positional arguments and which are optional keyword arguments, and to decide whether to use the advanced techniques of arbitrary argument passing.
  - The goal is to write Python functions that are:
    - Easy to read (the name and arguments need no explanations)
    - Easy to change (adding a new keyword argument does not break other parts of the code)
    - Easy to use

Python Programming Standards

- Returning values
  - When a function grows in complexity it is not uncommon to use multiple return statements inside the function's body
    - However, in order to keep a clear intent and a sustainable readability level, it is preferable to avoid returning meaningful values from many output points in the body.
  - There are two main cases for returning values in a function:
    - The result of the function return when it has been processed normally, and the error cases that indicate a wrong input parameter or any other reason for the function to not be able to complete its computation or task.

Python Programming Standards

- Returning values – be consistent

```
# Correct:

def foo(x):
    if x >= 0:
        return math.sqrt(x)
    else:
        return None

def bar(x):
    if x < 0:
        return None
    return math.sqrt(x)
```

```
# Wrong:

def foo(x):
    if x >= 0:
        return math.sqrt(x)

def bar(x):
    if x < 0:
        return
    return math.sqrt(x)
```

Python Programming Standards

- Conventions
  - Don't compare Boolean values to True or False using ==:

```
# Correct:
if greeting:
```

```
# Wrong:
if greeting == True:
```

Worse:

```
# Wrong:
if greeting is True:
```

Python Programming Standards

- Line Continuations
  - When a logical line of code is longer than the accepted limit, you need to split it over multiple physical lines.
  - The Python interpreter will join consecutive lines if the last character of the line is a backslash.

```
with open('/path/to/some/file/you/want/to/read') as file_1, \
     open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```

    - This should usually be avoided because of its fragility: a white space added to the end of the line, after the backslash, will break the code and may have unexpected results.

Python Programming Standards

- Line Continuations
  - A better solution is to use parentheses around the elements
    - This uses Python's implied line continuation inside parentheses, brackets and braces.
    - Long lines can be broken over multiple lines by wrapping expressions in parentheses.
    - Left with an unclosed parenthesis on an end-of-line the Python interpreter will join the next line until the parentheses are closed. The same behavior holds for curly and square braces.

Python Programming Standards

- Line Continuations
  - Using parenthesis forces the interpreter to join the next line(s).

```
# Correct:
# easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Python Programming Standards

- Caution
  - Python allows many tricks. A good example is that any client code can override an object's properties and methods: there is no "**private**" keyword in Python.
  - This doesn't mean that no properties are considered private, and that no proper encapsulation is possible in Python.
  - Rather, instead of relying on concrete walls erected by the developers around their code, the Python community prefers to rely on a set of conventions indicating that these elements should not be accessed directly.

Python Programming Standards

- The Goal
  - One of Guido's key insights is that code is read much more often than it is written.
  - The guidelines are intended to improve the readability of code and make it consistent across the wide spectrum of Python code.

Python Programming Standards