Names and Python Syntax

If we recall from previous lectures, Python syntax makes very few restrictions on the ways that we can name our variables, functions, and classes.

- Variables names must start with a letter or an underscore
- Every character after the first (if any) must be a letter, underscore, or number
- Names cannot be a reserved Python keyword:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

However, we have also discussed guidelines and naming *conventions*. These are not enforced by the language, but the community has agreed to abide by a standard that defines best practices.

Classifying Common Styles

Python is not the only language with naming conventions. To help us describe the naming conventions for Python's conventions, PEP 8 describes and names a set of common styles. The following naming styles are commonly distinguished:

- `b` (single lowercase letter)
- `B` (single uppercase letter)
- `lowercase`
- `lower_case_with_underscores`
- `UPPERCASE`
- `UPPER_CASE_WITH_UNDERSCORES`
- `CapitalizedWords` (or `CapWords`, or `CamelCase` – so named because of the bumpy look of its letters). This is also sometimes known as `StudlyCaps`.
  - When using abbreviations in `CapWords`, capitalize all the letters of the abbreviation. Thus `HTTPServerError` is better than `HttpServerError`.
- `mixedCase` (differs from `CapitalizedWords` by initial lowercase character!)
- `Capitalized_Words_With_Underscores` (ugly!)

We can write regular expressions that match each of these styles. Some expressions may match multiple styles. For example, a regex that matches `lowercase_with_underscores` will also match `lowercase`, `b`, and single-word `mixedCase` names. If want to match the "best" example, we should think about the order we check for matches.

Python-specific Naming Conventions

In addition to naming common styles, PEP 8 also discusses style guidelines specific to Python. It urges backwards compatibility with existing code, but it includes the following guidelines (and more):

- Class names should normally use the `CapWords` convention.
- Because exceptions should be classes, the class naming convention applies here. However, you should use the suffix ”Error” on your exception names (if the exception actually is an error).
- Function names should be `lowercase`, with words separated by underscores as necessary to improve readability
- Always use `self` for the first argument to instance methods.
- Use one leading underscore only for non-public methods and instance variables.
- Constants are usually defined on a module level and written in all capital letters with underscores separating words.
- Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability.
- “Magic” object or attributes that live in user-controlled namespaces use the `__double_leading_and_trailing_underscore__`

Regular Expression Examples

Styles

`b` : [a-z]

`B` : [A-Z]

`lowercase` : [a-z]+

`lower_case_with_underscores` : [a-z][a-z_]*

`UPPERCASE` : [A-Z]+

`UPPER_CASE_WITH_UNDERSCORES` : [A-Z][A-Z_]*

`CapitalizedWords` : [A-Z][a-zA-Z]*

`mixedCase` : [a-z][a-zA-Z]*

`Capitalized_Words_With_Underscores` : [A-Z][a-zA-Z_]*

Python Conventions

Class names : [A-Z][0-9a-zA-Z_]*

Exceptions : [A-Z][0-9a-zA-Z_]*Error

local variables : [a-z][0-9a-z_]*

Function names : [a-z][0-9a-z_]*\(

Instance variables (public) : self\.[a-z][a-z0-9_]*

Instance variables (private) : self\._[a-z_][a-z0-9_]*

Constants : [A-Z][A-Z0-9_]*

Modules : [a-z][0-9a-z_]0,15

"Magic" object or attribute : (__[a-z]__)|(__[a-z][a-z0-9_]*[a-z0-9]__)