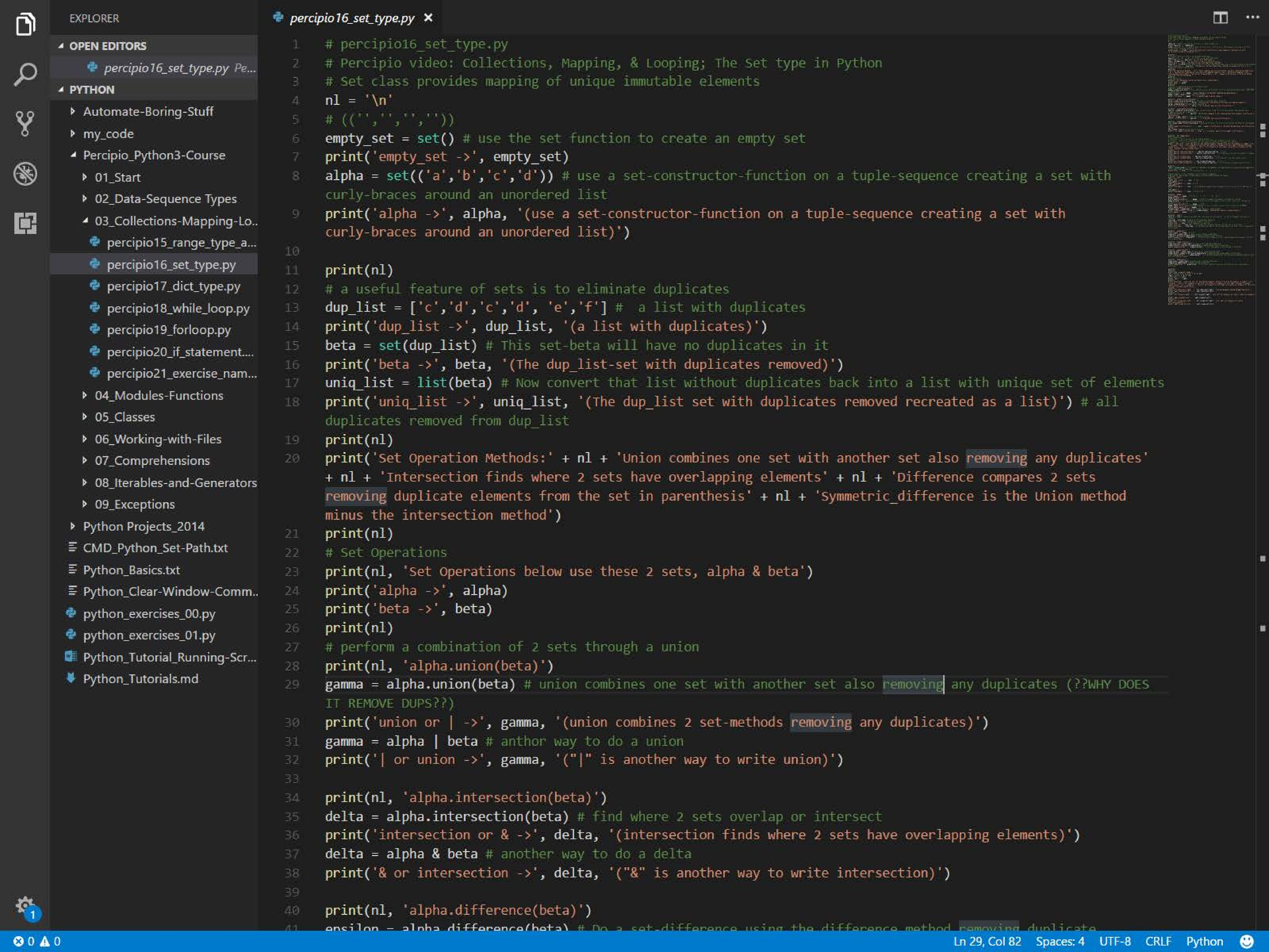
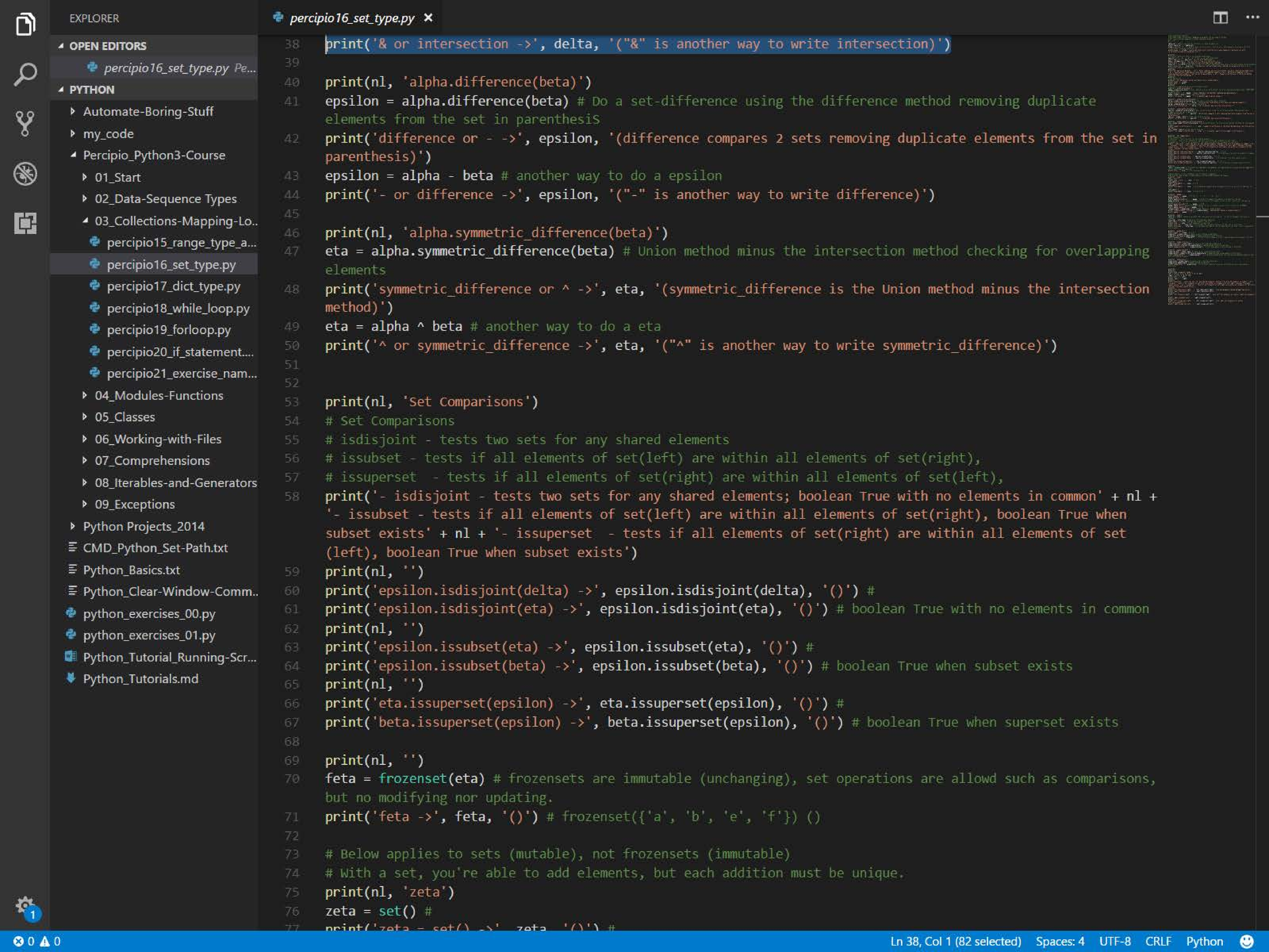
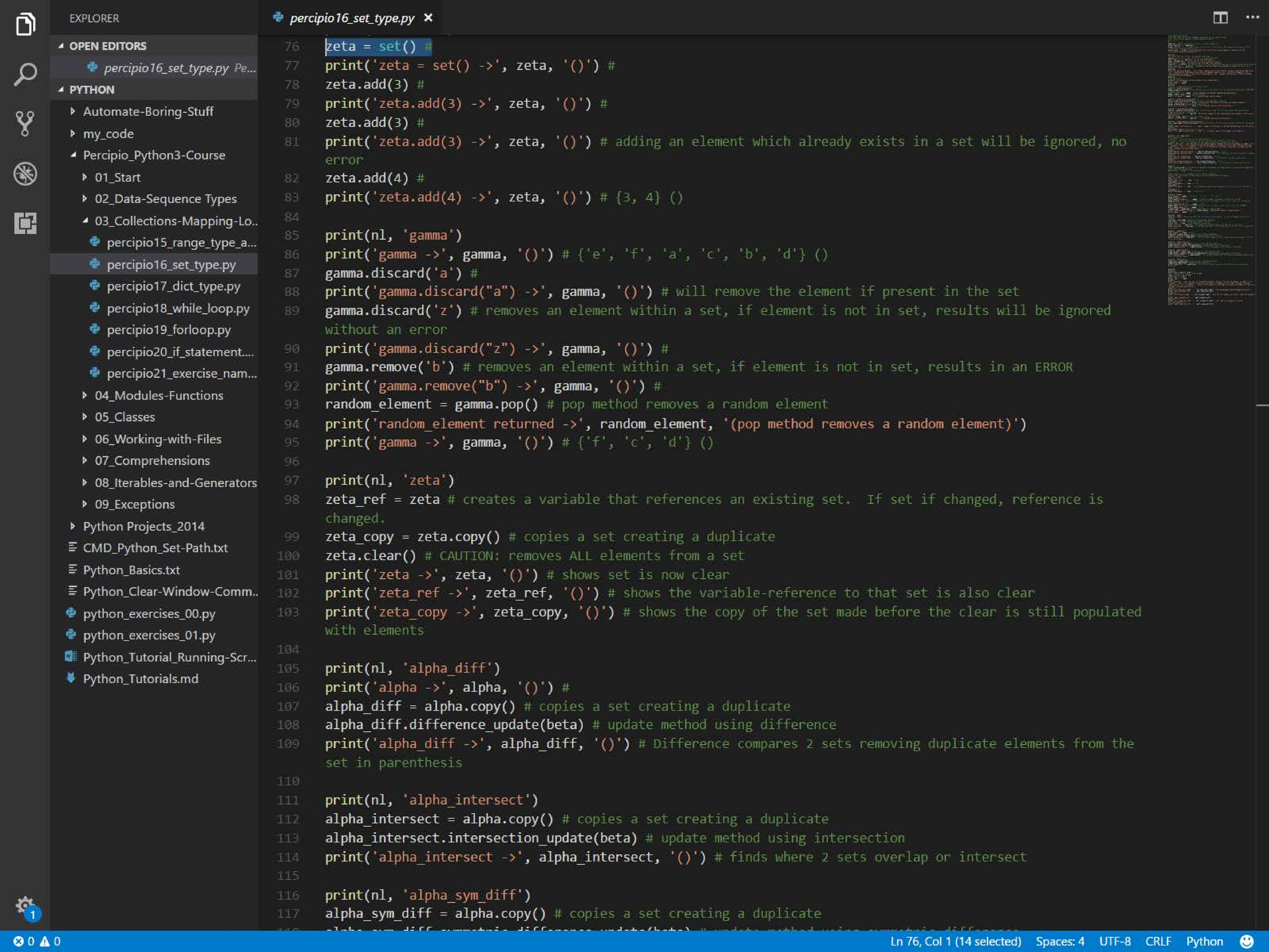


```python
# percipio16_set_type.py
# Percipio video: Collections, Mapping, & Looping; The Set type in Python
# Set class provides mapping of unique immutable elements
nl = '\n'
# (('','','',''))
empty_set = set() # use the set function to create an empty set
print('empty_set ->', empty_set)
alpha = set(('a','b','c','d')) # use a set-constructor-function on a tuple-sequence creating a set with curly-braces around an unordered list
print('alpha ->', alpha, '(use a set-constructor-function on a tuple-sequence creating a set with curly-braces around an unordered list)')

print(nl)
# a useful feature of sets is to eliminate duplicates
dup_list = ['c','d','c','d', 'e','f'] #  a list with duplicates
print('dup_list ->', dup_list, '(a list with duplicates)')
beta = set(dup_list) # This set-beta will have no duplicates in it
print('beta ->', beta, '(The dup_list-set with duplicates removed)')
uniq_list = list(beta) # Now convert that list without duplicates back into a list with unique set of elements
print('uniq_list ->', uniq_list, '(The dup_list set with duplicates removed recreated as a list)') # all duplicates removed from dup_list
print(nl)
print('Set Operation Methods:' + nl + 'Union combines one set with another set also removing any duplicates' + nl + 'Intersection finds where 2 sets have overlapping elements' + nl + 'Difference compares 2 sets removing duplicate elements from the set in parenthesis' + nl + 'Symmetric_difference is the Union method minus the intersection method')
print(nl)
# Set Operations
print(nl, 'Set Operations below use these 2 sets, alpha & beta')
print('alpha ->', alpha)
print('beta ->', beta)
print(nl)
# perform a combination of 2 sets through a union
print(nl, 'alpha.union(beta)')
gamma = alpha.union(beta) # union combines one set with another set also removing any duplicates (??WHY DOES IT REMOVE DUPS??)
print('union or | ->', gamma, '(union combines 2 set-methods removing any duplicates)')
gamma = alpha | beta # anthor way to do a union
print('| or union ->', gamma, '("|" is another way to write union)')

print(nl, 'alpha.intersection(beta)')
delta = alpha.intersection(beta) # find where 2 sets overlap or intersect
print('intersection or & ->', delta, '(intersection finds where 2 sets have overlapping elements)')
delta = alpha & beta # another way to do a delta
print('& or intersection ->', delta, '("&" is another way to write intersection)')

print(nl, 'alpha.difference(beta)')
epsilon = alpha.difference(beta) # Do a set-difference using the difference method removing duplicate
```

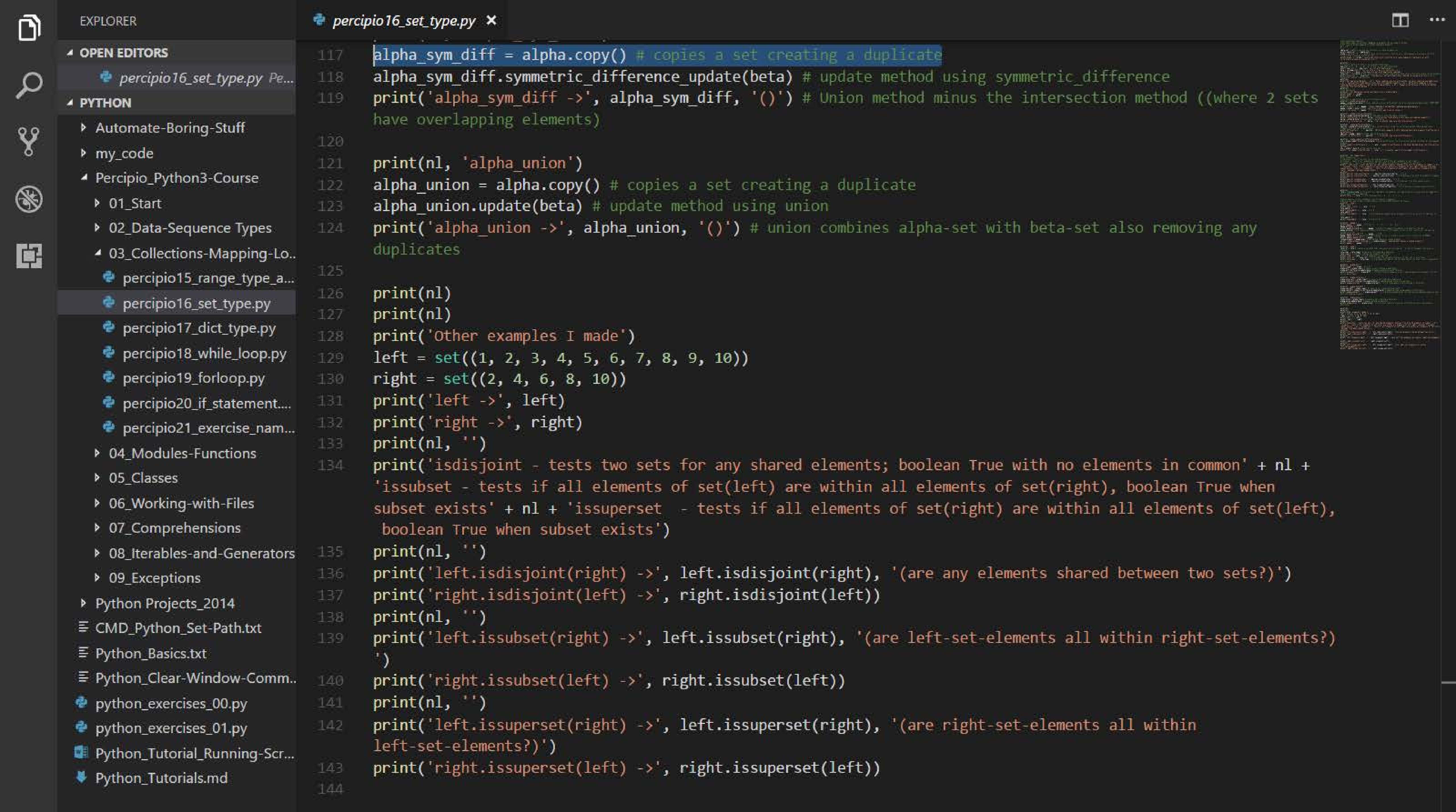


```python
print('& or intersection ->', delta, '("&" is another way to write intersection)')

print(nl, 'alpha.difference(beta)')
epsilon = alpha.difference(beta) # Do a set-difference using the difference method removing duplicate elements from the set in parenthesis
print('difference or - ->', epsilon, '(difference compares 2 sets removing duplicate elements from the set in parenthesis)')
epsilon = alpha - beta # another way to do a epsilon
print('- or difference ->', epsilon, '("-" is another way to write difference)')

print(nl, 'alpha.symmetric_difference(beta)')
eta = alpha.symmetric_difference(beta) # Union method minus the intersection method checking for overlapping elements
print('symmetric_difference or ^ ->', eta, '(symmetric_difference is the Union method minus the intersection method)')
eta = alpha ^ beta # another way to do a eta
print('^ or symmetric_difference ->', eta, '("^" is another way to write symmetric_difference)')


print(nl, 'Set Comparisons')
# Set Comparisons
# isdisjoint - tests two sets for any shared elements
# issubset - tests if all elements of set(left) are within all elements of set(right),
# issuperset  - tests if all elements of set(right) are within all elements of set(left),
print('- isdisjoint - tests two sets for any shared elements; boolean True with no elements in common' + nl + '- issubset - tests if all elements of set(left) are within all elements of set(right), boolean True when subset exists' + nl + '- issuperset  - tests if all elements of set(right) are within all elements of set(left), boolean True when subset exists')
print(nl, '')
print('epsilon.isdisjoint(delta) ->', epsilon.isdisjoint(delta), '()') #
print('epsilon.isdisjoint(eta) ->', epsilon.isdisjoint(eta), '()') # boolean True with no elements in common
print(nl, '')
print('epsilon.issubset(eta) ->', epsilon.issubset(eta), '()') #
print('epsilon.issubset(beta) ->', epsilon.issubset(beta), '()') # boolean True when subset exists
print(nl, '')
print('eta.issuperset(epsilon) ->', eta.issuperset(epsilon), '()') #
print('beta.issuperset(epsilon) ->', beta.issuperset(epsilon), '()') # boolean True when superset exists

print(nl, '')
feta = frozenset(eta) # frozensets are immutable (unchanging), set operations are allowd such as comparisons, but no modifying nor updating.
print('feta ->', feta, '()') # frozenset({'a', 'b', 'e', 'f'}) ()

# Below applies to sets (mutable), not frozensets (immutable)
# With a set, you're able to add elements, but each addition must be unique.
print(nl, 'zeta')
zeta = set() #
print('zeta = set() ->', zeta, '()') #
```

⊗ 0 ⚠ 0   Ln 38, Col 1 (82 selected)   Spaces: 4   UTF-8   CRLF   Python



```python
zeta = set() #
print('zeta = set() ->', zeta, '()') #
zeta.add(3) #
print('zeta.add(3) ->', zeta, '()') #
zeta.add(3) #
print('zeta.add(3) ->', zeta, '()') # adding an element which already exists in a set will be ignored, no error
zeta.add(4) #
print('zeta.add(4) ->', zeta, '()') # {3, 4} ()

print(nl, 'gamma')
print('gamma ->', gamma, '()') # {'e', 'f', 'a', 'c', 'b', 'd'} ()
gamma.discard('a') #
print('gamma.discard("a") ->', gamma, '()') # will remove the element if present in the set
gamma.discard('z') # removes an element within a set, if element is not in set, results will be ignored without an error
print('gamma.discard("z") ->', gamma, '()') #
gamma.remove('b') # removes an element within a set, if element is not in set, results in an ERROR
print('gamma.remove("b") ->', gamma, '()') #
random_element = gamma.pop() # pop method removes a random element
print('random_element returned ->', random_element, '(pop method removes a random element)')
print('gamma ->', gamma, '()') # {'f', 'c', 'd'} ()

print(nl, 'zeta')
zeta_ref = zeta # creates a variable that references an existing set.  If set if changed, reference is changed.
zeta_copy = zeta.copy() # copies a set creating a duplicate
zeta.clear() # CAUTION: removes ALL elements from a set
print('zeta ->', zeta, '()') # shows set is now clear
print('zeta_ref ->', zeta_ref, '()') # shows the variable-reference to that set is also clear
print('zeta_copy ->', zeta_copy, '()') # shows the copy of the set made before the clear is still populated with elements

print(nl, 'alpha_diff')
print('alpha ->', alpha, '()') #
alpha_diff = alpha.copy() # copies a set creating a duplicate
alpha_diff.difference_update(beta) # update method using difference
print('alpha_diff ->', alpha_diff, '()') # Difference compares 2 sets removing duplicate elements from the set in parenthesis

print(nl, 'alpha_intersect')
alpha_intersect = alpha.copy() # copies a set creating a duplicate
alpha_intersect.intersection_update(beta) # update method using intersection
print('alpha_intersect ->', alpha_intersect, '()') # finds where 2 sets overlap or intersect

print(nl, 'alpha_sym_diff')
alpha_sym_diff = alpha.copy() # copies a set creating a duplicate
alpha_sym_diff.symmetric_difference_update(beta) # update method using symmetric difference
```



```python
alpha_sym_diff = alpha.copy() # copies a set creating a duplicate
alpha_sym_diff.symmetric_difference_update(beta) # update method using symmetric_difference
print('alpha_sym_diff ->', alpha_sym_diff, '()') # Union method minus the intersection method ((where 2 sets have overlapping elements)

print(nl, 'alpha_union')
alpha_union = alpha.copy() # copies a set creating a duplicate
alpha_union.update(beta) # update method using union
print('alpha_union ->', alpha_union, '()') # union combines alpha-set with beta-set also removing any duplicates

print(nl)
print(nl)
print('Other examples I made')
left = set((1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
right = set((2, 4, 6, 8, 10))
print('left ->', left)
print('right ->', right)
print(nl, '')
print('isdisjoint - tests two sets for any shared elements; boolean True with no elements in common' + nl + 'issubset - tests if all elements of set(left) are within all elements of set(right), boolean True when subset exists' + nl + 'issuperset  - tests if all elements of set(right) are within all elements of set(left), boolean True when subset exists')
print(nl, '')
print('left.isdisjoint(right) ->', left.isdisjoint(right), '(are any elements shared between two sets?)')
print('right.isdisjoint(left) ->', right.isdisjoint(left))
print(nl, '')
print('left.issubset(right) ->', left.issubset(right), '(are left-set-elements all within right-set-elements?)')
print('right.issubset(left) ->', right.issubset(left))
print(nl, '')
print('left.issuperset(right) ->', left.issuperset(right), '(are right-set-elements all within left-set-elements?)')
print('right.issuperset(left) ->', right.issuperset(left))

```