```
# CMPT 145 Course material
# Copyright (c) 2017-2021 Michael C Horsch
# All rights reserved.
#
# This document contains resources for homework assigned to students of
# CMPT 145 and shall not be distributed without permission.  Posting this
# file to a public or private website, or providing this file to a person
# not registered in CMPT 145, constitutes Academic Misconduct, according
# to the University of Saskatchewan Policy on Academic Misconduct.
#
# Synopsis:
#    Assignment 7 solutions

import a7q8 as a7q8
import node as N

assert_report = "{}: {}; found '{}', expected '{}'"

################################################################################
###### testing to_string() #####
# empty node-chain
test_item = 'to_string()'
reason = 'Empty node chain'
input = None
expected = 'EMPTY'

result = a7q8.to_string(input)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

# node-chain size 1
input = N.node(1)
expected = '[ 1 | / ]'
reason = 'node chain with one node'

result = a7q8.to_string(input)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

# node-chain size 2
input = N.node(1, N.node('two'))
expected = '[ 1 | *-]-->[ two | / ]'
reason =  'node chain with two nodes'

result = a7q8.to_string(input)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

# node-chain size 3
input = N.node(1, N.node('two', N.node(3)))
expected =  '[ 1 | *-]-->[ two | *-]-->[ 3 | / ]'
reason = 'node chain with three nodes'

result = a7q8.to_string(input)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


################################################################################
```

```
#### testing: check_chains() ####
test_item = "check_chains()"

data_in1 = None
data_in2 = None
expected = True
reason = 'Two empty node chains'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

data_in1 = N.node(1)
data_in2 = None
expected = False
reason = 'One empty node chain'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


data_in1 = None
data_in2 = N.node(1)
expected = False
reason = 'One empty node chain'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


data_in1 = N.node(1)
data_in2 = N.node(1)
expected = True
reason = 'Simple node chains, same values'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


data_in1 = N.node(1)
data_in2 = data_in1
expected = True
reason = 'Simple node chain, two copies of the same reference'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


data_in1 = N.node(1, N.node(2, N.node(3)))
data_in2 = data_in1
expected = True
reason = 'longer node chain, two copies of the same reference'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
```

```
expected)

#### UNIT TEST CASE: check_chains() ####
data_in1 = N.node(1, N.node(2, N.node(3)))
data_in2 = N.node(1, N.node(2, N.node(3)))
expected = True
reason = 'longer node chain, two copies of the same chain'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

#### UNIT TEST CASE: check_chains() ####
data_in1 = N.node(1, N.node(2, N.node(3)))
data_in2 = N.node(1, N.node(2, N.node(1)))
expected = False
reason = 'longer node chain, similar, last value different'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


#### UNIT TEST CASE: check_chains() ####
test_item = 'check_chains()'

data_in1 = N.node(1, N.node(2, N.node(3)))
data_in2 = N.node(1, N.node(2, N.node(3, N.node(4))))
expected = False
reason = 'longer node chain, similar, second chain longer'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

#### UNIT TEST CASE: check_chains() ####
data_in1 = N.node(1, N.node(2, N.node(3, N.node(4))))
data_in2 = N.node(1, N.node(2, N.node(3)))
expected = False
reason = 'longer node chain, similar, first chain longer'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


#### UNIT TEST CASE: check_chains() ####
data_in1 = N.node(1, N.node(2, N.node(3, N.node(4))))
data_in2 = N.node(2, N.node(3, N.node(4)))
expected = False
reason = 'longer node chain, very different'

result = a7q8.check_chains(data_in1, data_in2)
assert result == expected, assert_report.format(test_item, reason, result,
expected)


################################################################################
###### testing copy() #####
```

```
# empty node-chain
test_item = 'copy()'
input =  None
expected = 'EMPTY'
reason = 'Empty node chain'

output = a7q8.copy(input)
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

# short node-chain
input = N.node(1)
expected = '[ 1 | / ]'
reason = 'copying a short node-chain'

output = a7q8.copy(input)
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
assert input is not output, 'copy(): returned the original node-chain'

#longer node-chain
input =  N.node(5 ,N.node(10,N.node(-15, N.node(1))))
expected = '[ 5 | *-]-->[ 10 | *-]-->[ -15 | *-]-->[ 1 | / ]'
reason = 'copying a longer node-chain'

output = a7q8.copy(input)
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
assert input is not output, 'copy(): returned the original node-chain'




################################################################################
###### testing replace() #####
# empty node-chain
test_item = 'replace()'
inputs =  [None, 1, 1]
expected = "EMPTY"
reason = 'Empty node chain'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)

# short node-chain with one replacement
input_chain = N.node(1)
inputs = [input_chain, 1, 2]
expected = "[ 2 | / ]"
reason = 'node chain with one node, target in'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
```

```
assert input_chain is output, 'replace(): did not return the original node-chain'

# short node-chains no replacements
# test1
input_chain = N.node(1)
inputs = [input_chain, 5, 2]
expected = "[ 1 | / ]"
reason = 'node chain with one node, target not in'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
assert input_chain is output, 'replace(): did not return the original node-chain'

# test2
input_chain = N.node(1, N.node(2))
inputs = [input_chain, 5, 2]
expected = "[ 1 | *-]-->[ 2 | / ]"
reason = 'node chain with two nodes, target not in'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
assert input_chain is output, 'replace(): did not return the original node-chain'

# short node-chain replacement at start
input_chain = N.node('one', N.node(2))
inputs = [input_chain, 'one', 1]
expected = "[ 1 | *-]-->[ 2 | / ]"
reason = 'node chain with two nodes, target last'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
assert input_chain is output, 'replace(): did not return the original node-chain'

# short node-chain replacement at end
input_chain = N.node(1, N.node('two'))
inputs = [input_chain, 'two', 2]
expected = "[ 1 | *-]-->[ 2 | / ]"
reason = 'node chain with two nodes, target last'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
assert result == expected, assert_report.format(test_item, reason, result,
expected)
assert input_chain is output, 'replace(): did not return the original node-chain'

# short node-chain replacement in middle
input_chain = N.node(1, N.node('two', N.node(3)))
inputs = [input_chain, 'two', 2]
expected = "[ 1 | *-]-->[ 2 | *-]-->[ 3 | / ]"
reason = 'node chain with three nodes, target middle'

output = a7q8.replace(inputs[0], inputs[1], inputs[2])
result = a7q8.to_string(output)
```

```
    assert result == expected, assert_report.format(test_item, reason, result,
    expected)
    assert input_chain is output, 'replace(): did not return the original node-chain'
```