30.4.2023 12:30:17 **graph_test.py** Page 1/1

```python

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path   : uebung10/al/aufgabe02
# Version: Sun Apr 30 12:30:17 CEST 2023

import sys
from uebung10.al.aufgabe02.graph import Graph


if __name__ == '__main__':

  graph = Graph()
  u = graph.insert_vertex("U")
  v = graph.insert_vertex("V")
  w = graph.insert_vertex("W")
  a = graph.insert_edge(u, v, "a")
  b = graph.insert_edge(v, w, "b")
  graph.print_graph()

  if graph.opposite(u, a) is not v:
    print("ERROR: v is not opposite of u!")
    sys.exit(1)
  if not graph.are_adjacent(v, w):
    print("ERROR: v is not adjacent of w!")
    sys.exit(2)


""" Session-Log:

Graph:
U -> (V,a)
V -> (U,a)(W,b)
W -> (V,b)

"""
```

30.4.2023 12:30:17 **graph.py** Page 1/2

```python

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path   : uebung10/al/aufgabe02
# Version: Sun Apr 30 12:30:17 CEST 2023

from uebung10.graphs.graph_impl import GraphImpl

class Graph:

  def __init__(self):
    self._graph = GraphImpl()

  def num_vertices(self):
    return self._graph.vertex_count()

  def num_edges(self):
    return self._graph.edge_count()

  def vertices(self):
    return self._graph.vertices()

  def edges(self):
    return self._graph.edges()

  def replace_in_vertex(self, v, element):
    self._validate_vertex(v)
    return v.replace(element)

  def replace_in_edge(self, e, element):
    self._validate_edge(e)
    return e.replace(element)

  def incident_edges(self, v):
    self._validate_vertex(v)
    return self._graph.incident_edges(v)

  def end_vertices(self, e):
    self._validate_edge(e)
    end_vertices = e.endpoints()
    return (end_vertices[0], end_vertices[1])

  def opposite(self, v, e):
    """
    Raises a ValueError if edge is not incident to vertex.
    """
    self._validate_vertex(v)
    self._validate_edge(e)

    # TODO: Implement here ...

    return None

  def are_adjacent(self, v, w):
    self._validate_vertex(v)
    self._validate_vertex(w)
    inc_v = self.incident_edges(v)
    inc_w = self.incident_edges(w)

    # TODO: Implement here ...

    return None

  def insert_vertex(self, element):
    v = self._graph.insert_vertex(element)
    return v

  def insert_edge(self, v, w, element):
    self._validate_vertex(v)
    self._validate_vertex(w)
    e = self._graph.insert_edge(v, w, element)
    return e
```

```
72
73    def remove_vertex(self, v):
74      edges = list(self.incident_edges(v))
75      for e in edges:
76        self.remove_edge(e)
77      del self._graph._outgoing[v]
78      return v.element()
79
80    def remove_edge(self, e):
81      self._validate_edge(e)
82      end_vertices = self.end_vertices(e)
83      del self._graph._outgoing[end_vertices[0]][end_vertices[1]]
84      del self._graph._outgoing[end_vertices[1]][end_vertices[0]]
85      return e.element()
86
87    def _validate_vertex(self, v):
88      return self._graph._validate_vertex(v)
89
90    def _validate_edge(self, e):
91      return self._graph._validate_edge(e)
92
93    def print_graph(self):
94      print("Graph:")
95      for v in self._graph.vertices():
96        print(str(v), end = " -> ")
97        for e in self.incident_edges(v):
98          w = self.opposite(v, e)
99          print("(" + str(w) + ",", end = "")
100         print(str(e.element()) + ")", end = "")
101       print("")
102     print("")
103
104
```