Q1

(a) The integer value for a and b will result in TypeError before the execution of line 8, hence no execution of the fault.

Input: a=0 b=1

Expected Output: TypeError

Actual Output: TypeError

(b) b is not the right type, so when executing line 8, q=len(b), it executes the fault, and the output is TypeError(q=undefined, p1=undefined), and the program terminates. However, if we fix line 8, the internal state is still the same, so we do execute the fault, but the internal state is still the same, hence no error.

Input: a= [[0, 0], [0, 0]] b=1

Expected Output: TypeError

Actual Output: TypeError

(c) When execution of line 8, the internal state is from "q=2, p1=undefined" to "q=2, p1=2", but if we fix line8, the internal state is from "p1=2, q=undefined" to "p1=2, q=2", it is obvious that internal state changes, so it causes an error. However, as for the two programs, after line b, their internal states are same. As a result, there is no failure.

Input: a= [[0, 0], [0, 0]] b= [[1, 1], [1, 1]]

Expected Output: [[0, 0], [0, 0]]

Actual Output: [[0, 0], [0, 0]]

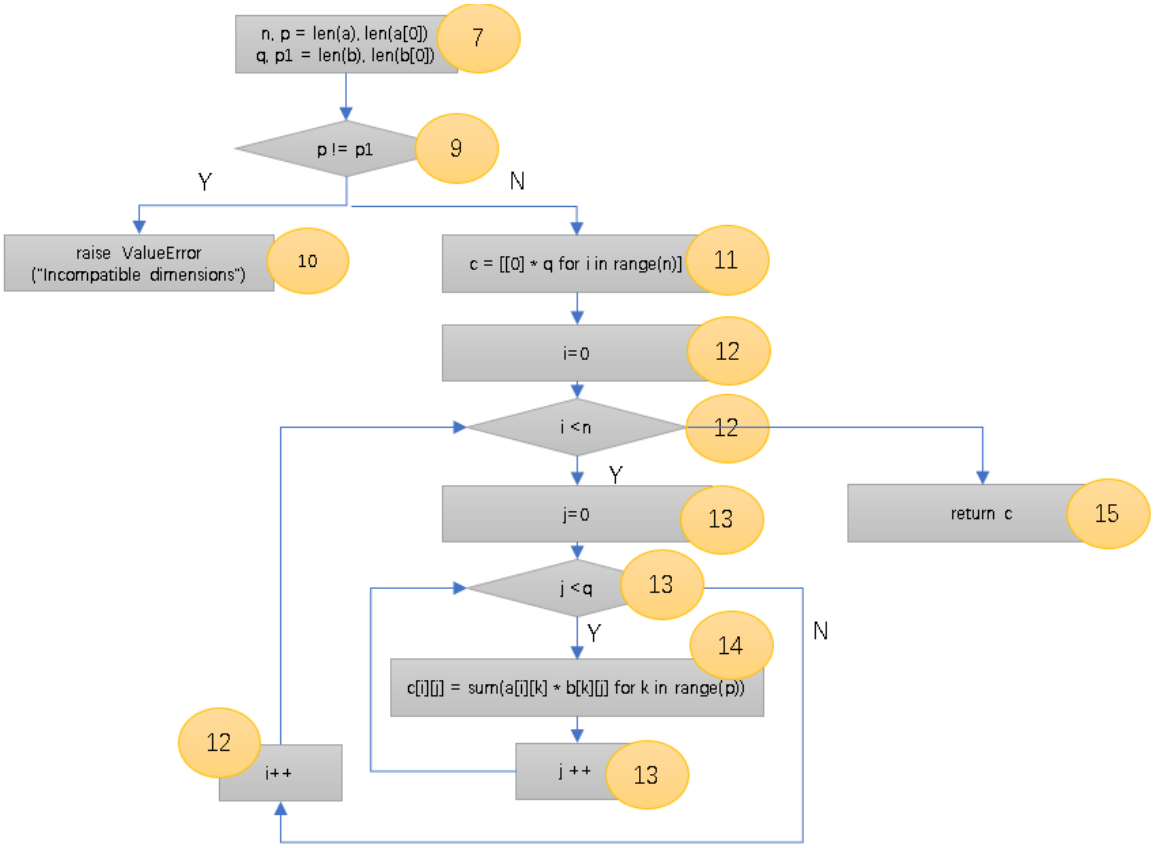
(d) First error state:

a = [[5, 7], [8, 21]], b = [[8], [4]] n=2 p=2 q=2 p1= undefined pc = q= len(b)

Complete state :

a = [[5, 7], [8, 21]], b = [[8], [4]] n=2 p=2 q=2 p1=1 pc= raise ValueError("Impossible dimensions")

(e)



Q2

(a)

```
class AST(object):
    pass

class Stmt (ASt):
    pass

class RepeatUntilStmt(object):
    def __init__(self,repeat_stmt,cond):
        self.repear_stmt=repeat_stmt
        self.cond=cond
```

(b)

$$\frac{< b, q >\Downarrow true, < S, q >\Downarrow q'}{< repeat\ S\ until\ b, q >\Downarrow q'}$$

$$\frac{< b, q >\Downarrow false, < S, q >\Downarrow q', < repeat\ S\ until\ b, q' >\Downarrow q''}{< repeat\ S\ until\ b, q >\Downarrow q''}$$

(c)
$$\frac{<x:=2;[]>\Downarrow[x:=2] \quad ,<0,[]>\Downarrow 0}{<x\leq 0;[x:=2]>\Downarrow false}$$

Since it has been proved that

$$\frac{< b,q >\Downarrow false,< S,q >\Downarrow q',< repeat\ S\ until\ b,q' >\Downarrow q''}{< repeat\ S\ until\ b,q >\Downarrow q''}$$

$$\frac{< x \leq 0;[x := 2] >\Downarrow false\ ,\ < x := x-1,[x := 2] >\Downarrow [x := 1]\ ,\quad < repeat\ x := x-1\ until\ x \leq 0,[x := 1] >\Downarrow [x := 0]}{< repeat\ x := x-1\ until\ x \leq 0,[x := 2] >\Downarrow [x := 0]}$$

So it is obvious that ⍰x := 2 ; repeat x := x − 1 until x ≤ 0, [ ]⍰ ⇓ [x := 0]

(d)First, we prove repeat S until b⟹ S ; if b then skip else (repeat S until b)

When<b, q>⇓true

$$\frac{< b,q >\Downarrow true,< S,q >\Downarrow q'}{< repeat\ S\ until\ b,q >\Downarrow q'}$$

$$\frac{< b,q >\Downarrow true,< skip,q >\Downarrow q'}{< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q'}$$

$$\frac{< b,q >\Downarrow true,< s,q >\Downarrow q',< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q'}{< S; if\ b\ then\ \ skip\ else(repeat\ S\ until\ b) >\Downarrow q'}$$

When<b, q>⇓false

$$\frac{< b,q >\Downarrow false,< S,q >\Downarrow q',< repeat\ S\ until\ b,q >\Downarrow q''}{< repeat\ S\ until\ b,q >\Downarrow q''}$$

$$\frac{< b,q >\Downarrow false,< repeat\ S\ until\ b,q' > q''}{< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q''}$$

$$\frac{< b,q >\Downarrow false,< s,q >\Downarrow q',< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b),q' >\Downarrow q''}{< S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q''}$$

So, we prove repeat S until b⟹ S ; if b then skip else (repeat S until b)

Then, we prove repeat S until b⟸ S ; if b then skip else (repeat S until b)

When<b, q>⇓true

$$\frac{< S,q >\Downarrow q',< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q''}{< S;\ if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q''}$$

$$\frac{< skip,q' >\Downarrow q''}{< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b),q' >\Downarrow q''}$$

$$\frac{< S,q >\Downarrow q'}{< repeat\ S\ until\ b,q >\Downarrow q' = q''}$$

When<b, q>⇓false

$$\frac{< S,q >\Downarrow q',< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q''}{< S;\ if\ b\ then\ skip\ else\ (repeat\ S\ until\ b) >\Downarrow q''}$$

$$\frac{< repeat\ S\ until\ b,q' >\Downarrow q''}{< if\ b\ then\ skip\ else\ (repeat\ S\ until\ b),q' >\Downarrow q''}$$

$$\frac{< repeat\ S\ until\ b,q' >\Downarrow q''}{< repeat\ S\ until\ b,q >\Downarrow q''}$$

So, we prove repeat S until b⟸ S ; if b then skip else (repeat S until b)

Above all, repeat S until b and S ; if b then skip else (repeat S until b) are semantically equivalent.

Q3

(c)
On the original programme, self.assertEqual('one^|uno||three^^^^|four^^^|^cuatro|'),['one|uno', '', 'three^^', 'four^|cuatro',''] is true.
However, as for mutation1 and 2, the result is false. So it is obvious that the test kills both mutants, so the test case is useful.
As for mutation 1, on line 28, I changed "+=" to "=", so we can see that variables are assigned different values, the mutant may be killed.
As for mutation 2, on line 13, I changed "==" to "!=", so we can see that when the program has a different logic expression, the mutant may be killed.

Q4

(a)
ast.py:38: AST cannot be instantiated, so we cannot reach line 38.
ast.py:378-379: node.is_unary() is always false, since unary is not available in the program
ast.py:398: StmtList is always larger than None, so we cannot reach line 398.

int.py:35: __repr__ returned non-string, so we cannot reach line 38.
int.py:75: the possible values of 'op' have been used in the judgement of if, so we cannot reach line 75
int.py:155: if assert statement is not ture, there will be failure in the test.
int.py:179-184; 188-193; 197: these lines are reachable only when we run int.py file.

parser.py: 118;269: Only when the statement is not in the right format, we can reach them. But this will cause error.
parser.py: 452-453: the regular expression of Newline is wrong, so we cannot reach them.
parse.py:561-569;573-582: these lines are reachable only when we run parser.py file.

(b)
ast.py:377: node.is_unary() is always false, since unary is not available in the program
ast.py:397: StmtList is always larger than None, so we cannot reach the branch

int.py:72: the possible values of 'op' have been used in the judgement of if, so we cannot reach the branch

int.py:90: line 90 is never false, so we cannot reach the branch
int.py:111: line 111 is never false, so we cannot reach the branch
int.py:155: if assert statement is not ture, there will be failure in the test.
Int.py:196: the branch is reachable only when we run int.py file.

parser.py:67: keywords is never None, so we cannot reach the branch
parser.py:572: this branch cannot be reached until we run parser.py file