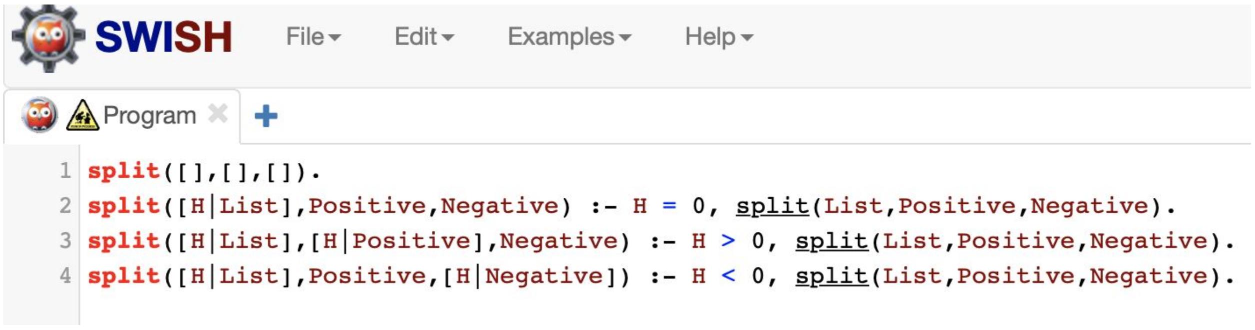
1. Write a Prolog program to split a list into two lists of positive and negative numbers.

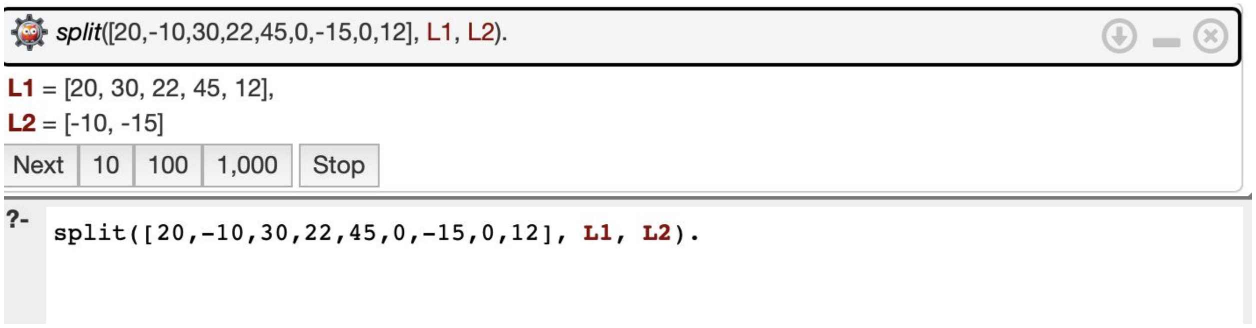
Actual Code:



```
split([],[],[]).
split([H|List],Positive,Negative) :- H = 0, split(List,Positive,Negative).
split([H|List],[H|Positive],Negative) :- H > 0, split(List,Positive,Negative).
split([H|List],Positive,[H|Negative]) :- H < 0, split(List,Positive,Negative).
```

The code provided uses the head and tail functionality of prolog. As you can see there are different variations of how the head is used. With the head and tails, the head takes the very beginning of the list. As we see in the output, L1 takes 20 as it is the first number to appear. However in L2, we see that -10 is chosen first. The reason for this, is that we are creating two list, one positive(L1), and one negative(L2). If we look in the query, -10 is the second number in the list, however it's the first negative integer. Afterwards, for L1 takes the tail, or remaining list that are positive and places them in accordance with L1. The same follows for L2, however with the negative numbers.

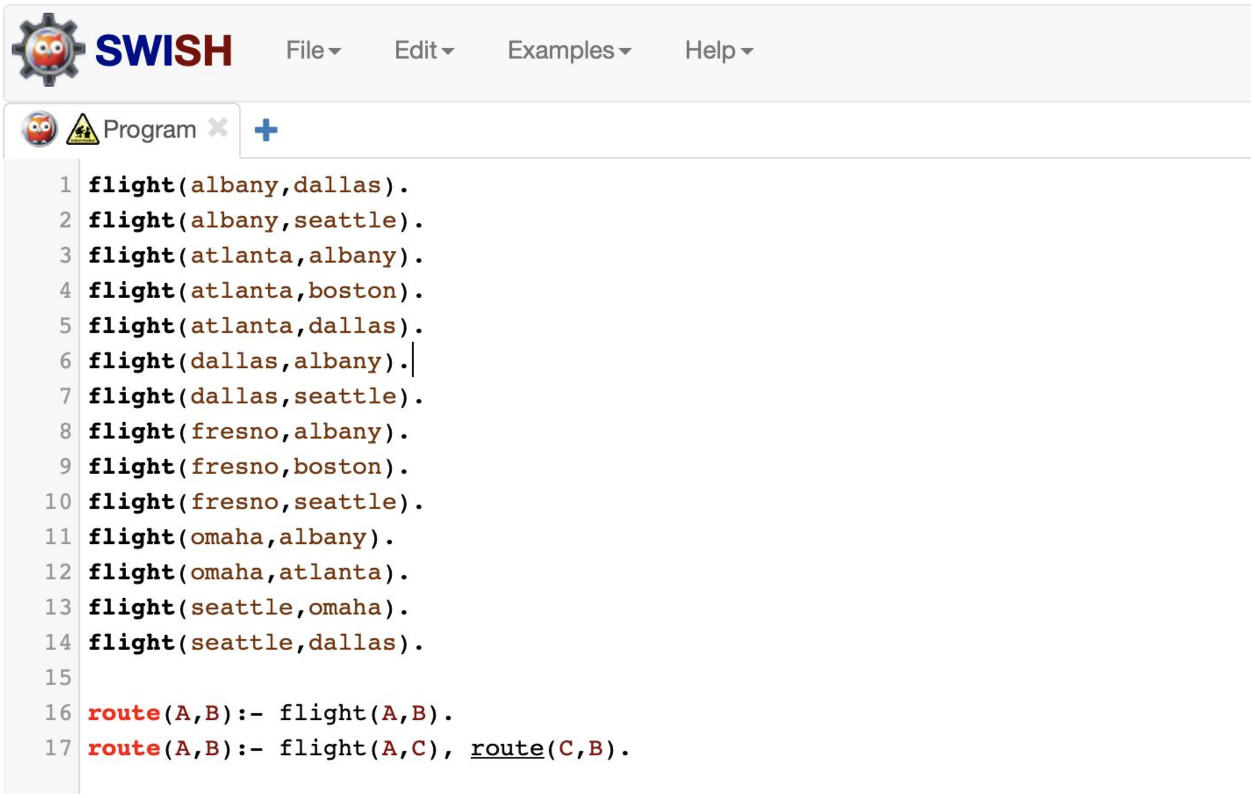
Output:



```
split([20,-10,30,22,45,0,-15,0,12], L1, L2).

L1 = [20, 30, 22, 45, 12],
L2 = [-10, -15]

?- split([20,-10,30,22,45,0,-15,0,12], L1, L2).
```

2. Given following graph of possible flights between seven US cities:

Actual Code



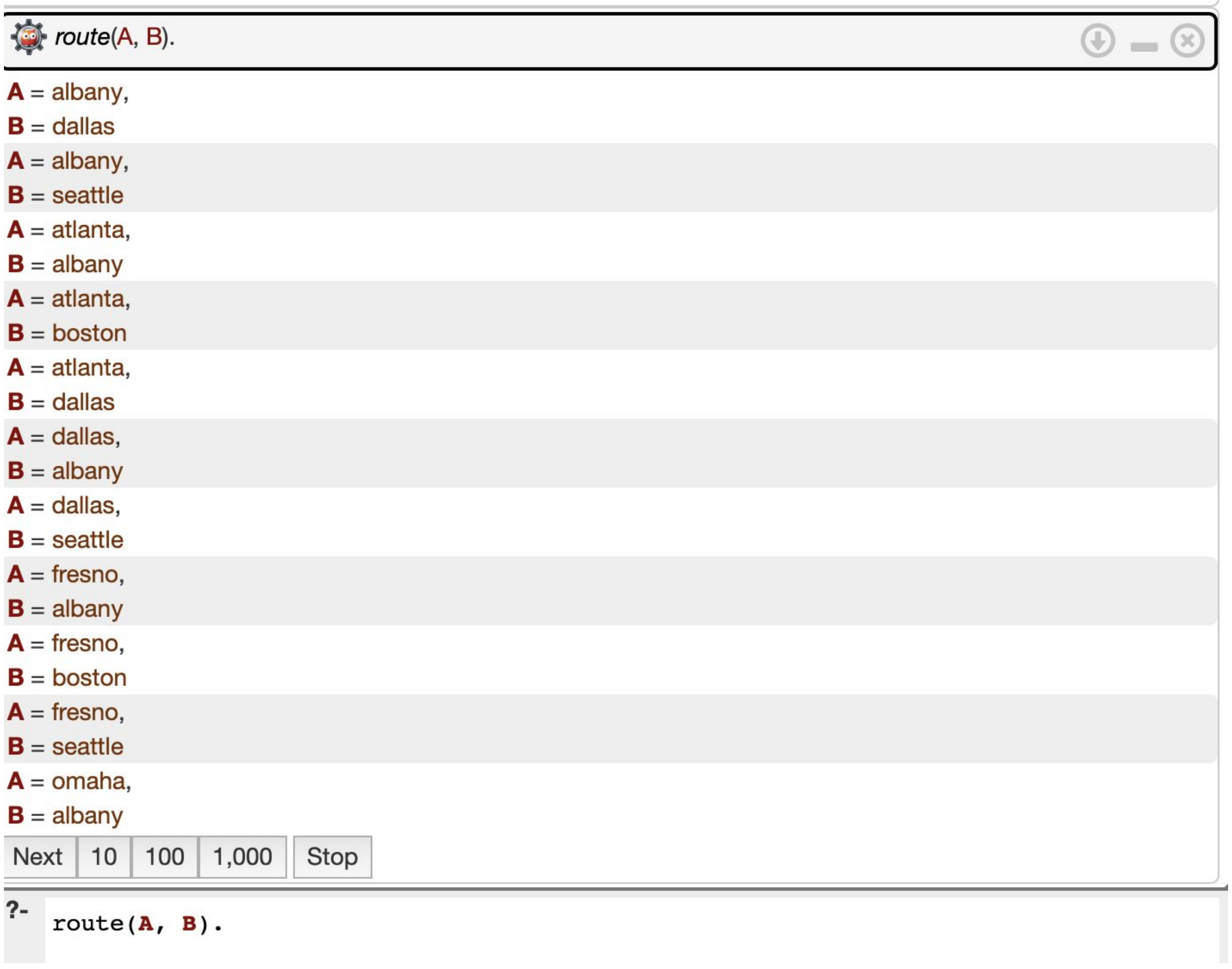
```prolog
flight(albany,dallas).
flight(albany,seattle).
flight(atlanta,albany).
flight(atlanta,boston).
flight(atlanta,dallas).
flight(dallas,albany).
flight(dallas,seattle).
flight(fresno,albany).
flight(fresno,boston).
flight(fresno,seattle).
flight(omaha,albany).
flight(omaha,atlanta).
flight(seattle,omaha).
flight(seattle,dallas).

route(A,B):- flight(A,B).
route(A,B):- flight(A,C), route(C,B).
```

Output with samples provided:

What does it mean that there is a route from city A to city B? Well, we either have a direct connection, or we can to go from A to some city from which we have a direct connection to B. This is written using recursion as:

    route(A, B) :- ???
    route(A, B) :- ???

When we state that there is a route(A,B) :- this means that when asking the output in the query, the question is only true if the statement after :- is true. This :- stands for if and only if. For my example, i set route(A,B) if and only if the flight(A,B) matches. Below, the screenshot shows the flight plan A and B.

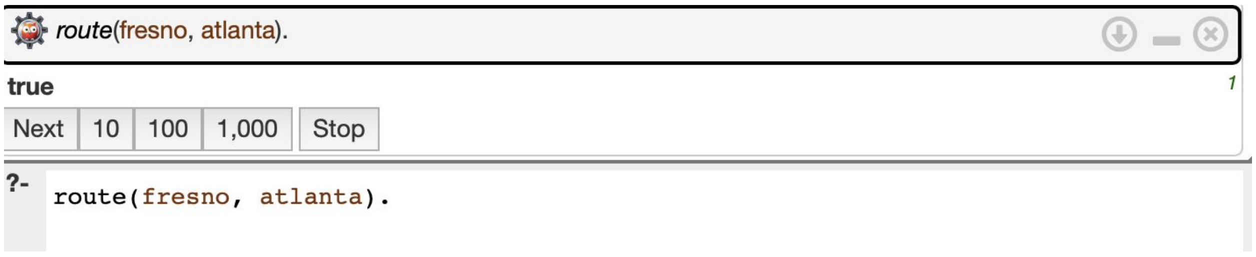


```
route(A, B).
A = albany,
B = dallas
A = albany,
B = seattle
A = atlanta,
B = albany
A = atlanta,
B = boston
A = atlanta,
B = dallas
A = dallas,
B = albany
A = dallas,
B = seattle
A = fresno,
B = albany
A = fresno,
B = boston
A = fresno,
B = seattle
A = omaha,
B = albany
```

```
?- route(A, B).
```

Now you can ask Prolog if there is a routs from a city to a city, for example:

```
    ?- route(fresno, atlanta).
```

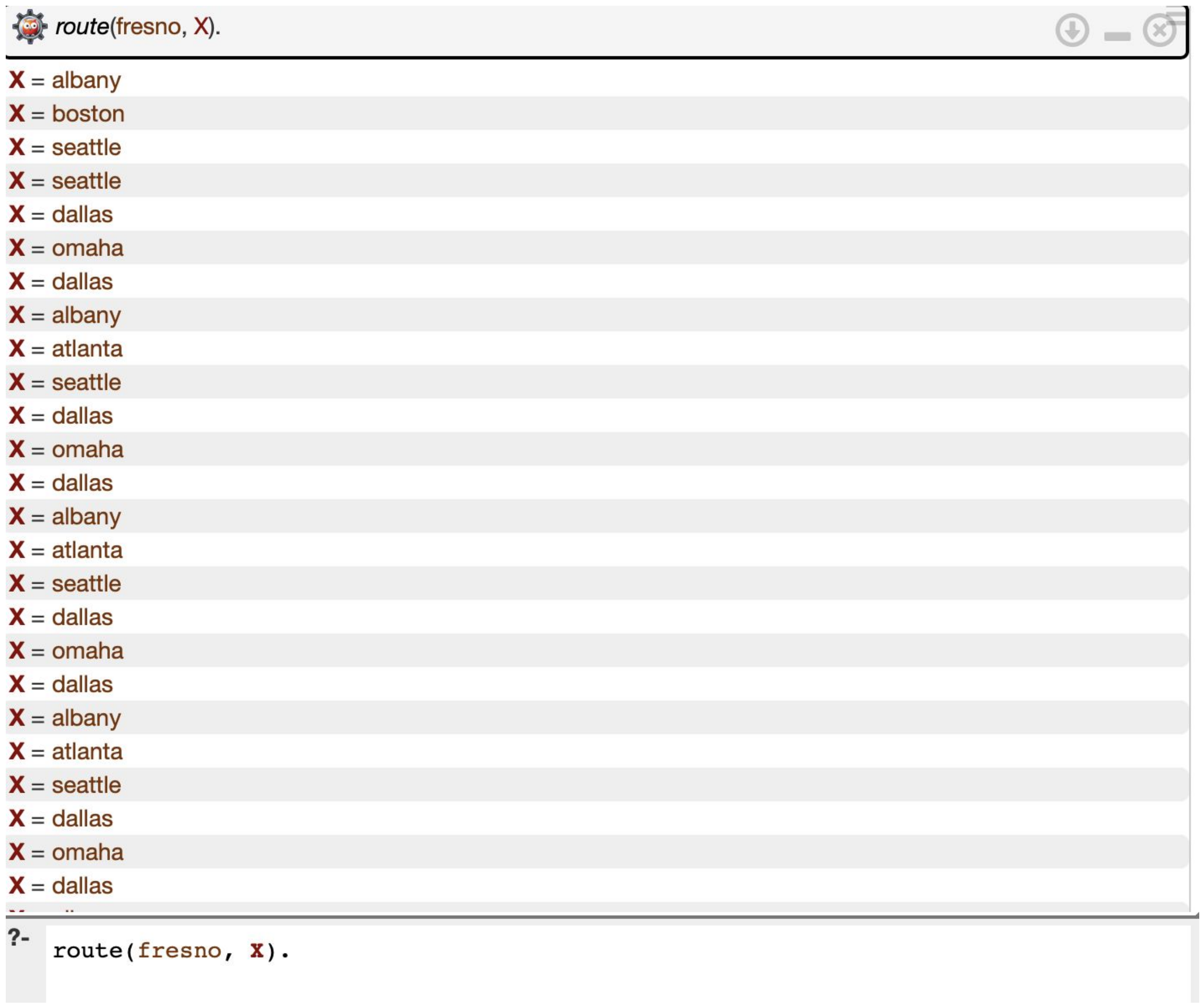


```
route(fresno, atlanta).
true                                                        1
```

```
?- route(fresno, atlanta).
```

You can ask all possible flights originated from a city, for example:

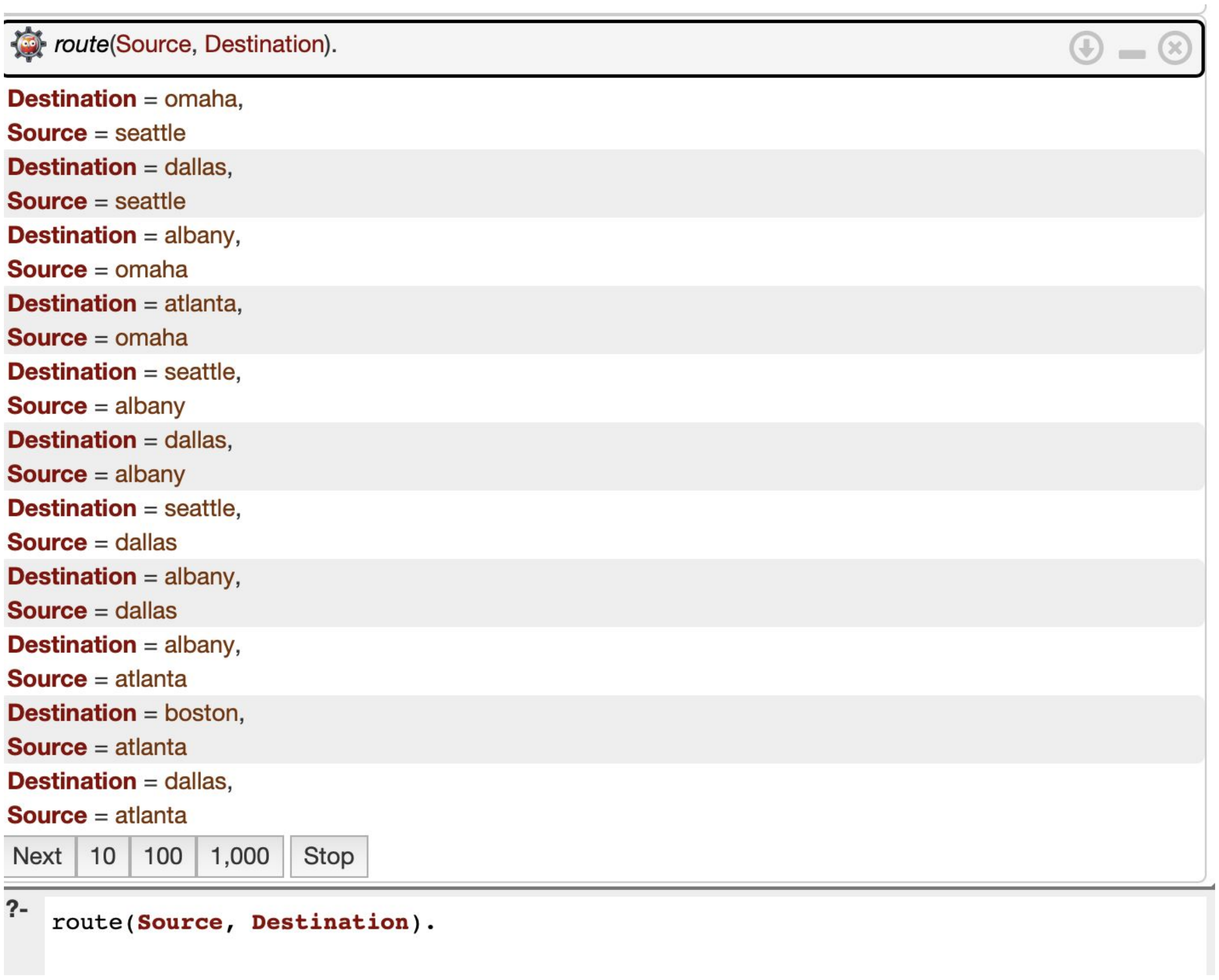
```
?- route(fresno, X).
```



```
route(fresno, X).
X = albany
X = boston
X = seattle
X = seattle
X = dallas
X = omaha
X = dallas
X = albany
X = atlanta
X = seattle
X = dallas
X = omaha
X = dallas
X = albany
X = atlanta
X = seattle
X = dallas
X = omaha
X = dallas
X = albany
X = atlanta
X = seattle
X = dallas
X = omaha
X = dallas
?- route(fresno, X).
```

You can also find out every possible flight, for example:

```
    ?- route(Source, Destination).
```



```
route(Source, Destination).
Destination = omaha,
Source = seattle
Destination = dallas,
Source = seattle
Destination = albany,
Source = omaha
Destination = atlanta,
Source = omaha
Destination = seattle,
Source = albany
Destination = dallas,
Source = albany
Destination = seattle,
Source = dallas
Destination = albany,
Source = dallas
Destination = albany,
Source = atlanta
Destination = boston,
Source = atlanta
Destination = dallas,
Source = atlanta
```

```
?- route(Source, Destination).
```