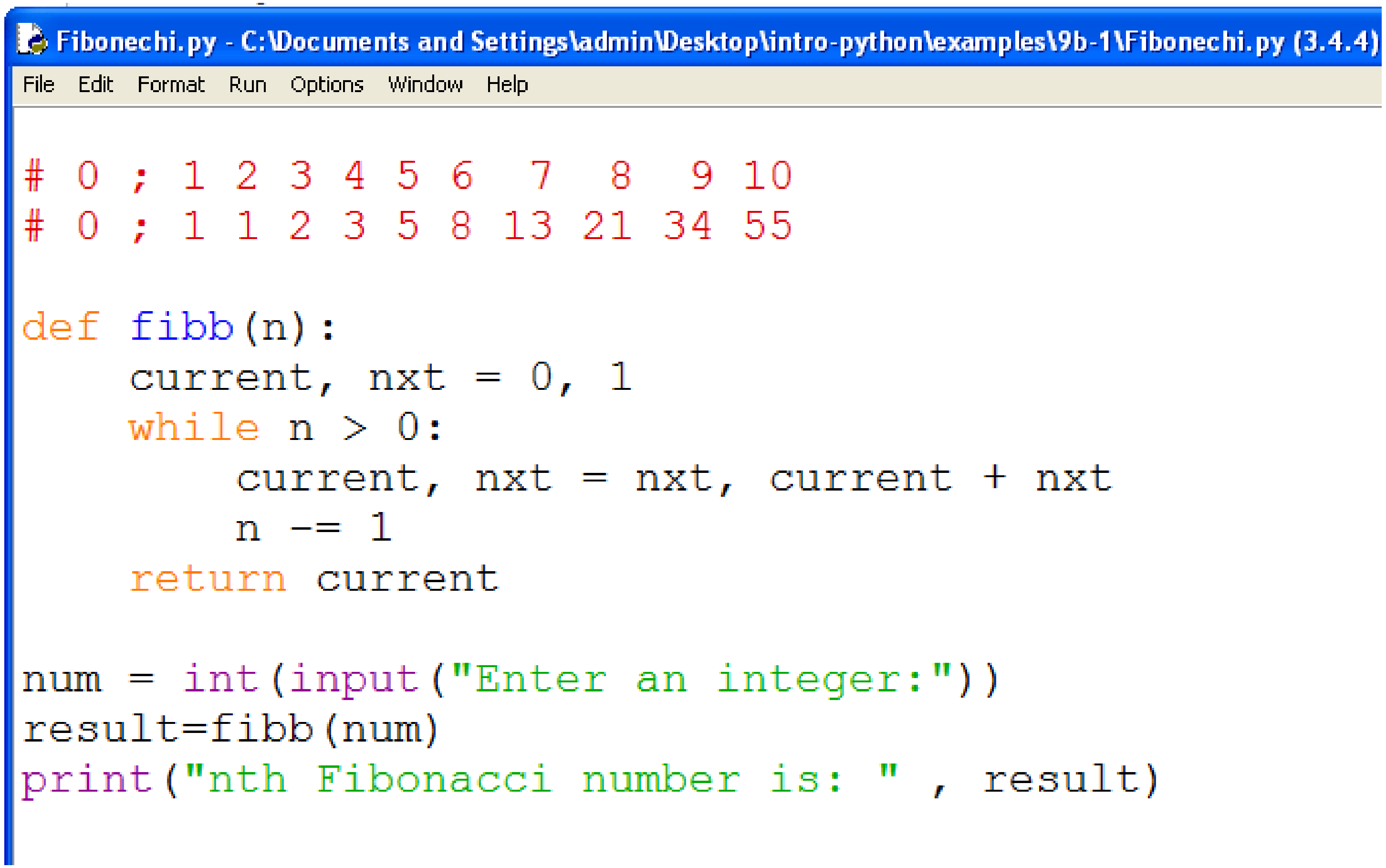
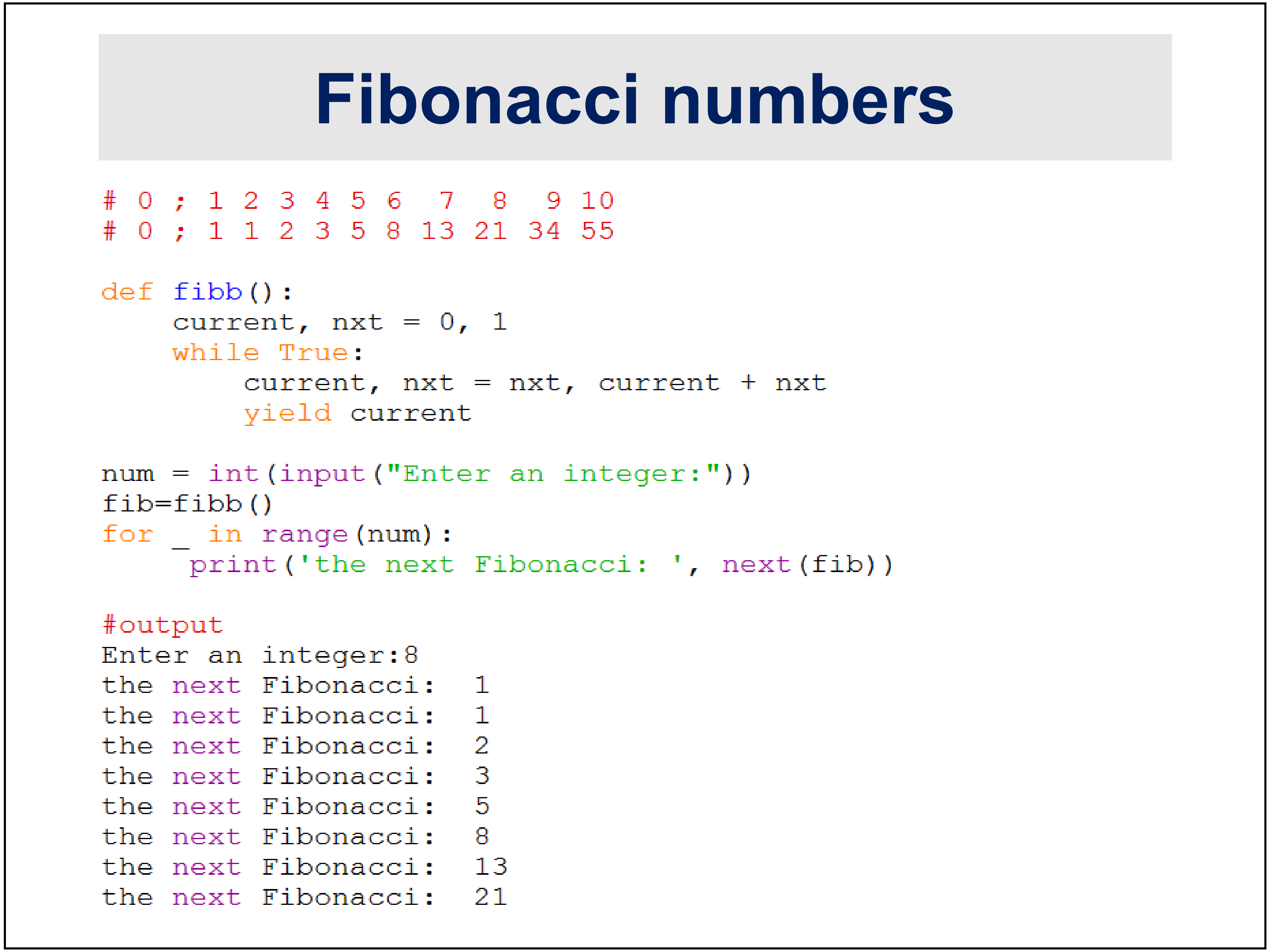
Fibonacci numbers


Fibonechi.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\9b-1\Fibonechi.py (3.4.4)
File Edit Format Run Options Window Help

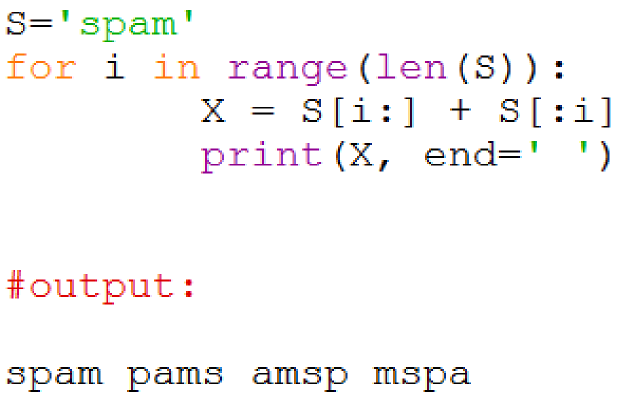
```
# 0 ; 1 2 3 4 5 6  7  8  9 10
# 0 ; 1 1 2 3 5 8 13 21 34 55

def fibb(n):
    current, nxt = 0, 1
    while n > 0:
        current, nxt = nxt, current + nxt
        n -= 1
    return current

num = int(input("Enter an integer:"))
result=fibb(num)
print("nth Fibonacci number is: " , result)
```



Fibonacci numbers

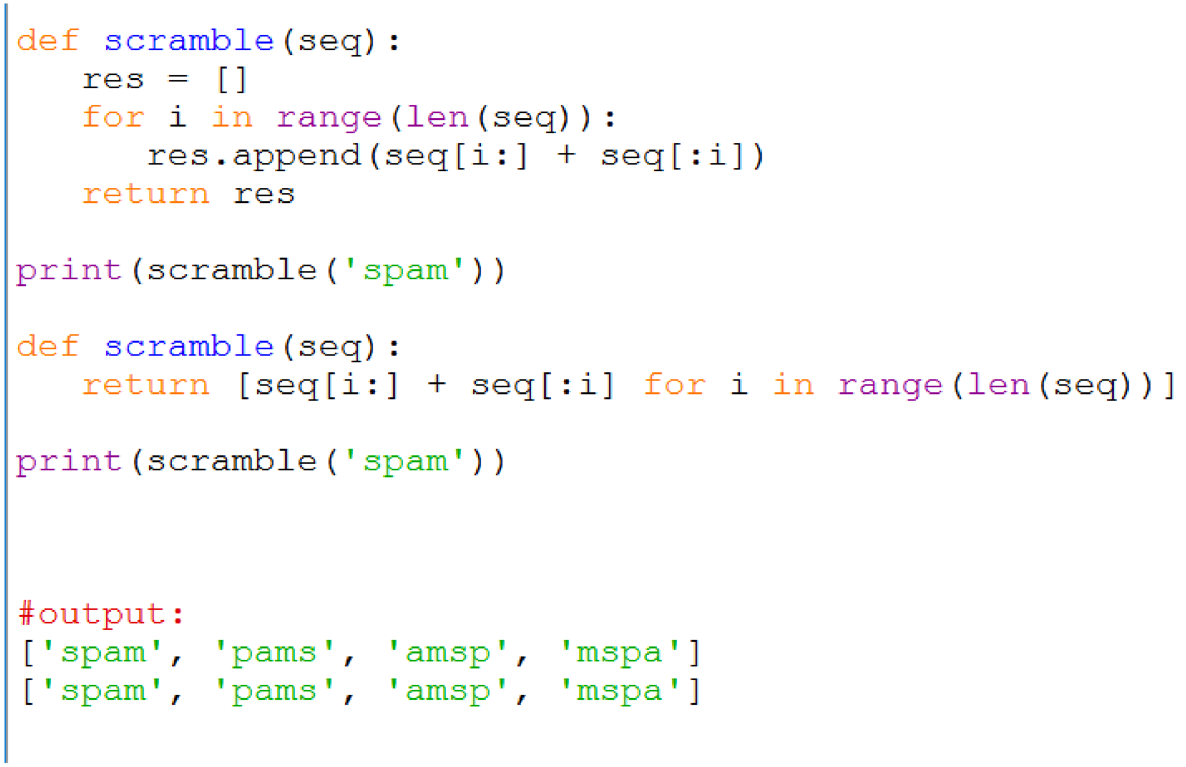
```
# 0 ; 1 2 3 4 5 6  7  8  9 10
# 0 ; 1 1 2 3 5 8 13 21 34 55

def fibb():
    current, nxt = 0, 1
    while True:
        current, nxt = nxt, current + nxt
        yield current

num = int(input("Enter an integer:"))
fib=fibb()
for _ in range(num):
     print('the next Fibonacci: ', next(fib))

#output
Enter an integer:8
the next Fibonacci:  1
the next Fibonacci:  1
the next Fibonacci:  2
the next Fibonacci:  3
the next Fibonacci:  5
the next Fibonacci:  8
the next Fibonacci:  13
the next Fibonacci:  21
```

Generating Scrambled Sequences



```
S='spam'
for i in range(len(S)):
        X = S[i:] + S[:i]
        print(X, end=' ')


#output:

spam pams amsp mspa
```

Generating Scrambled Sequences



```
def scramble(seq):
    res = []
    for i in range(len(seq)):
        res.append(seq[i:] + seq[:i])
    return res

print(scramble('spam'))

def scramble(seq):
    return [seq[i:] + seq[:i] for i in range(len(seq))]

print(scramble('spam'))



#output:
['spam', 'pams', 'amsp', 'mspa']
['spam', 'pams', 'amsp', 'mspa']
```

Generating Scrambled Sequences

```
def scramble(seq):
    for i in range(len(seq)):
        yield seq[i:] + seq[:i]

print(list(scramble('spam')))

def scramble(seq):
    return (seq[i:] + seq[:i] for i in range(len(seq)))

print(list(scramble('spam')))

#output:

['spam', 'pams', 'amsp', 'mspa']
['spam', 'pams', 'amsp', 'mspa']
```

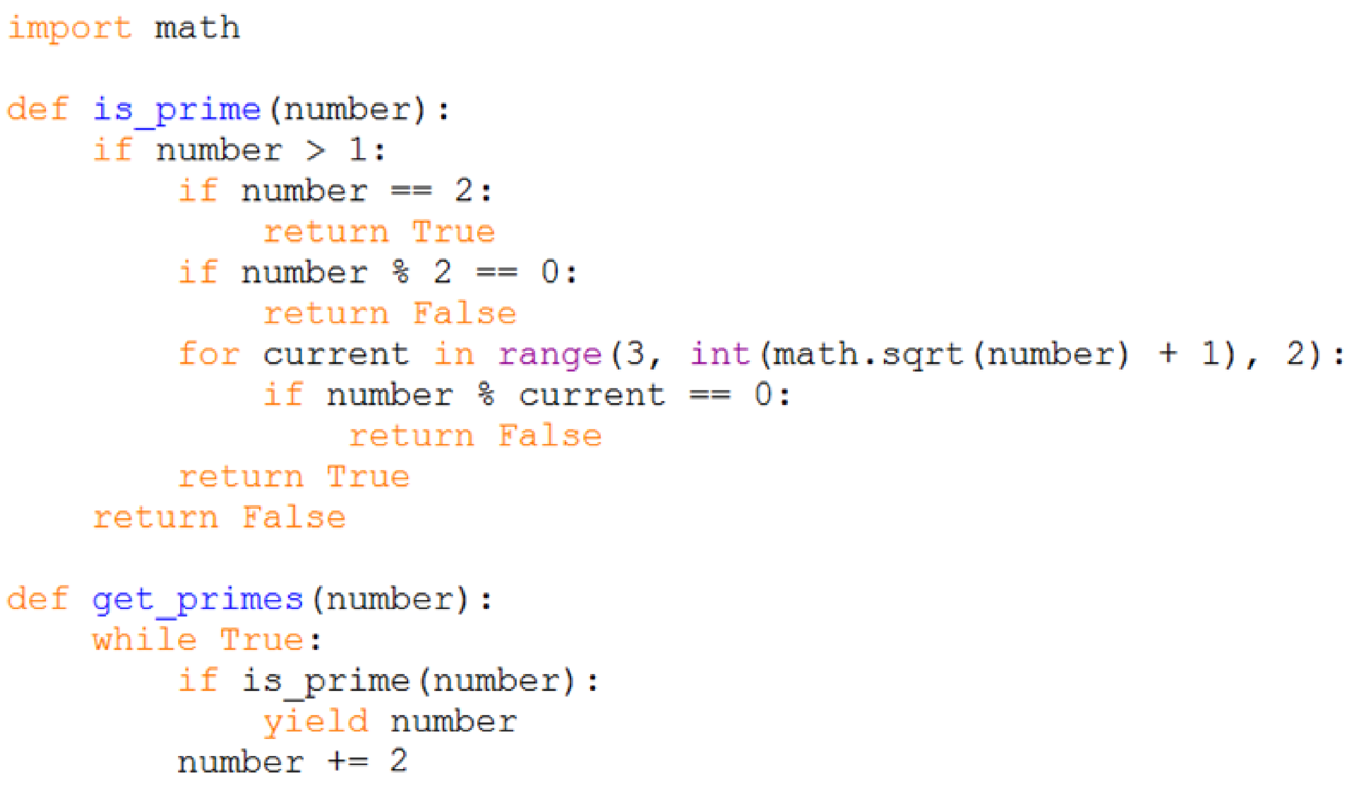
Merge

```
def merge(left, right):
     llen, rlen, i, j = len(left), len(right), 0, 0
     while i < llen or j < rlen:
          if j == rlen or (i < llen and left[i] < right[j]):
               yield left[i]
               i += 1
          else:
               yield right[j]
               j += 1

def printM(g):
   while True:
        try:
           print (next(g))
        except (StopIteration):
           print ("Done")
           break
```

Merge

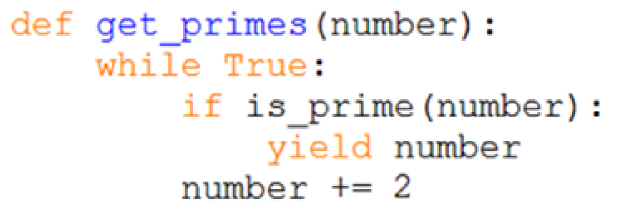
```
n=int(input("Enter the lenght of first array: "))
print("enter the data for first array: ", end='')
a=list(map(int,input().strip().split(" ")))

m=int(input("Enter the lenght of second array: "))
print("enter the data for second  array: ", end='')
b=list(map(int,input().strip().split(" ")))

g = merge(a, b)
printM(g)

#output
Enter the lenght of first array: 2
enter the data for first array: 1 2
Enter the lenght of second array: 3
enter the data for second  array: 2 5 8
1
2
2
5
8
Done
```

Adding Primes

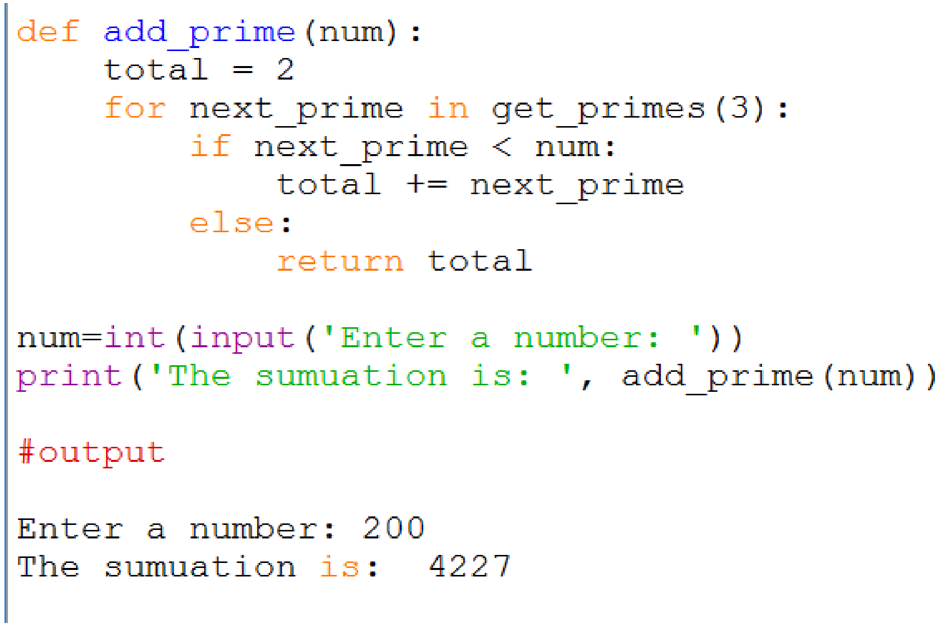


```python
import math

def is_prime(number):
    if number > 1:
        if number == 2:
            return True
        if number % 2 == 0:
            return False
        for current in range(3, int(math.sqrt(number) + 1), 2):
            if number % current == 0:
                return False
        return True
    return False

def get_primes(number):
    while True:
        if is_prime(number):
            yield number
        number += 2
```

Adding Primes

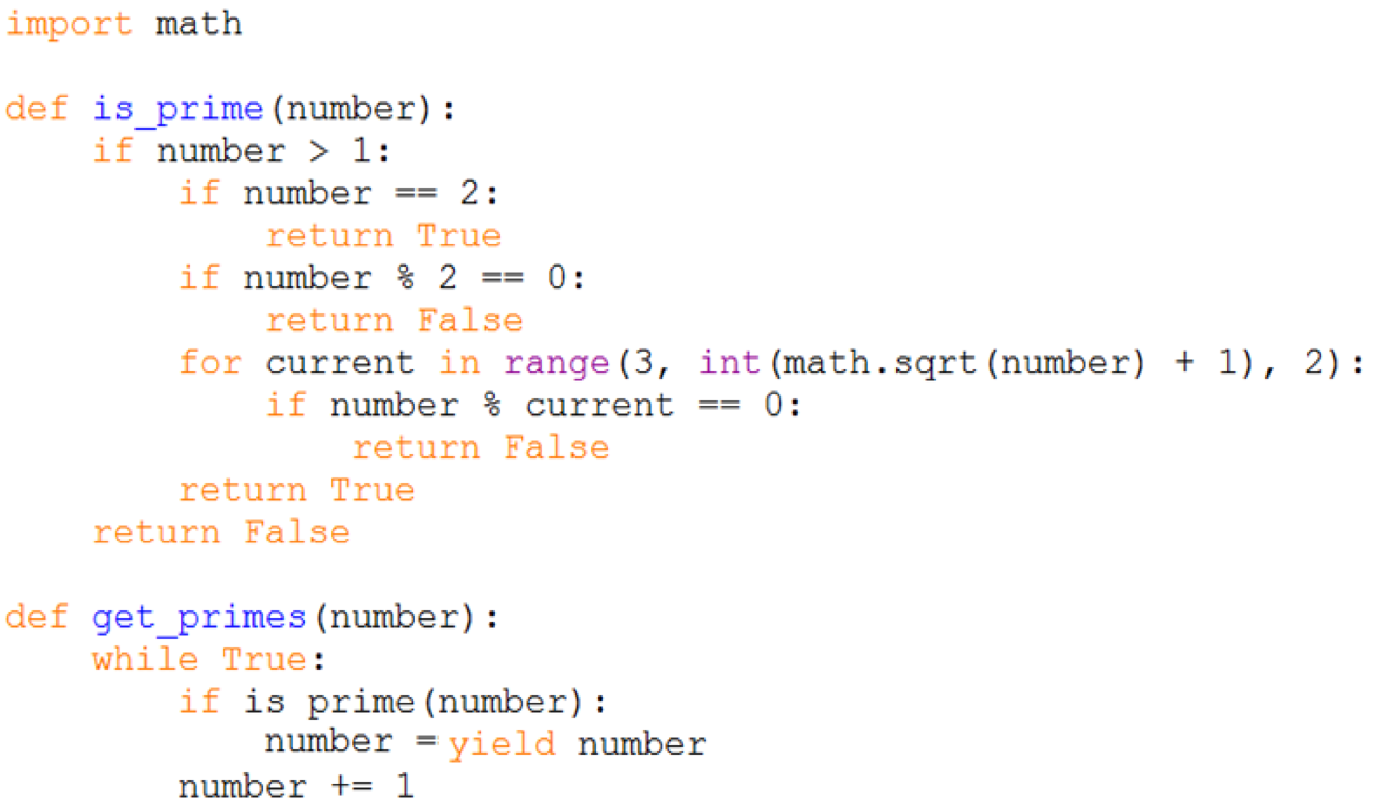


```
def get_primes(number):
    while True:
        if is_prime(number):
            yield number
        number += 2
```

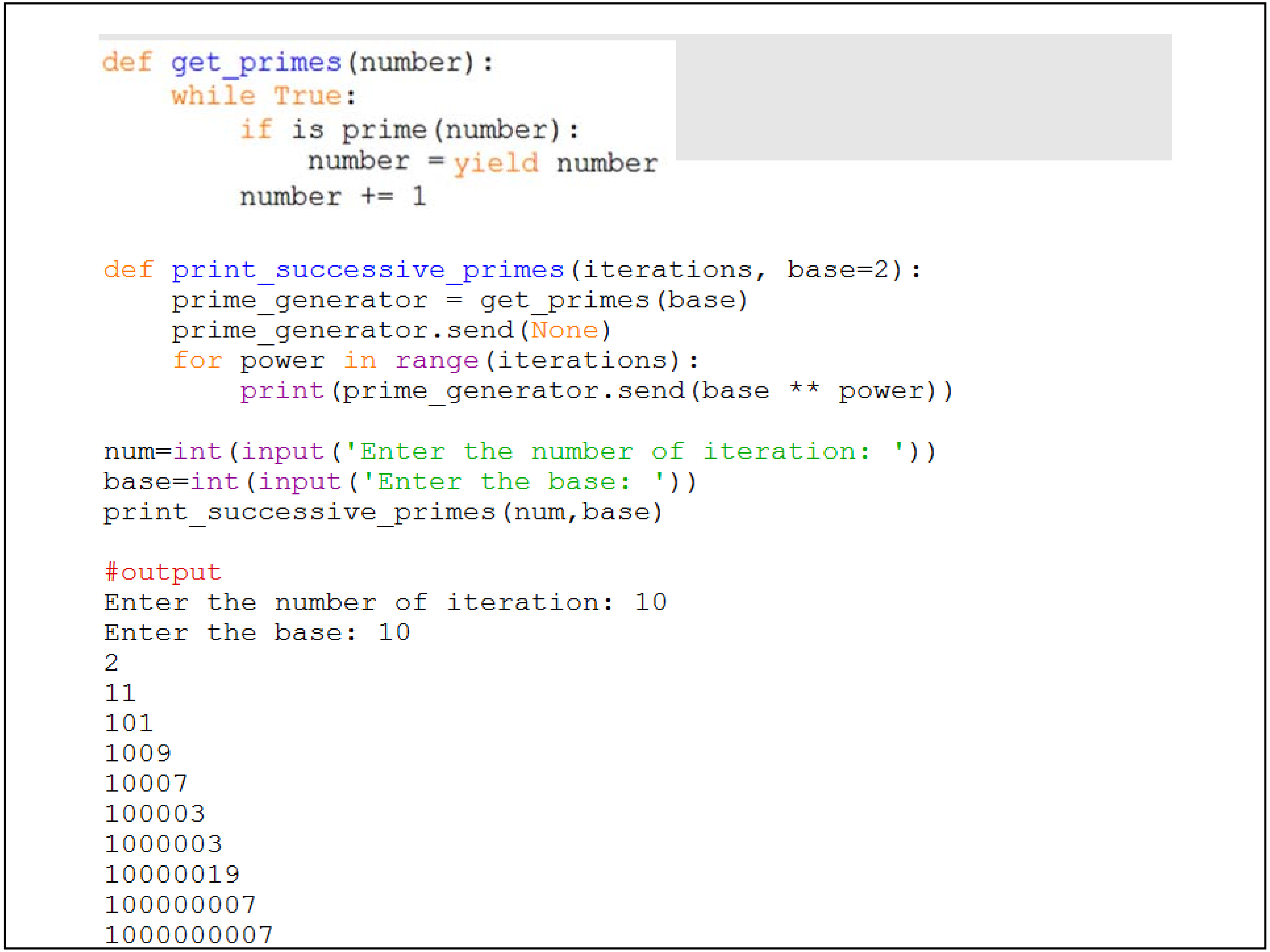


```
def add_prime(num):
    total = 2
    for next_prime in get_primes(3):
        if next_prime < num:
            total += next_prime
        else:
            return total

num=int(input('Enter a number: '))
print('The sumuation is: ', add_prime(num))

#output

Enter a number: 200
The sumuation is:  4227
```

Power Primes

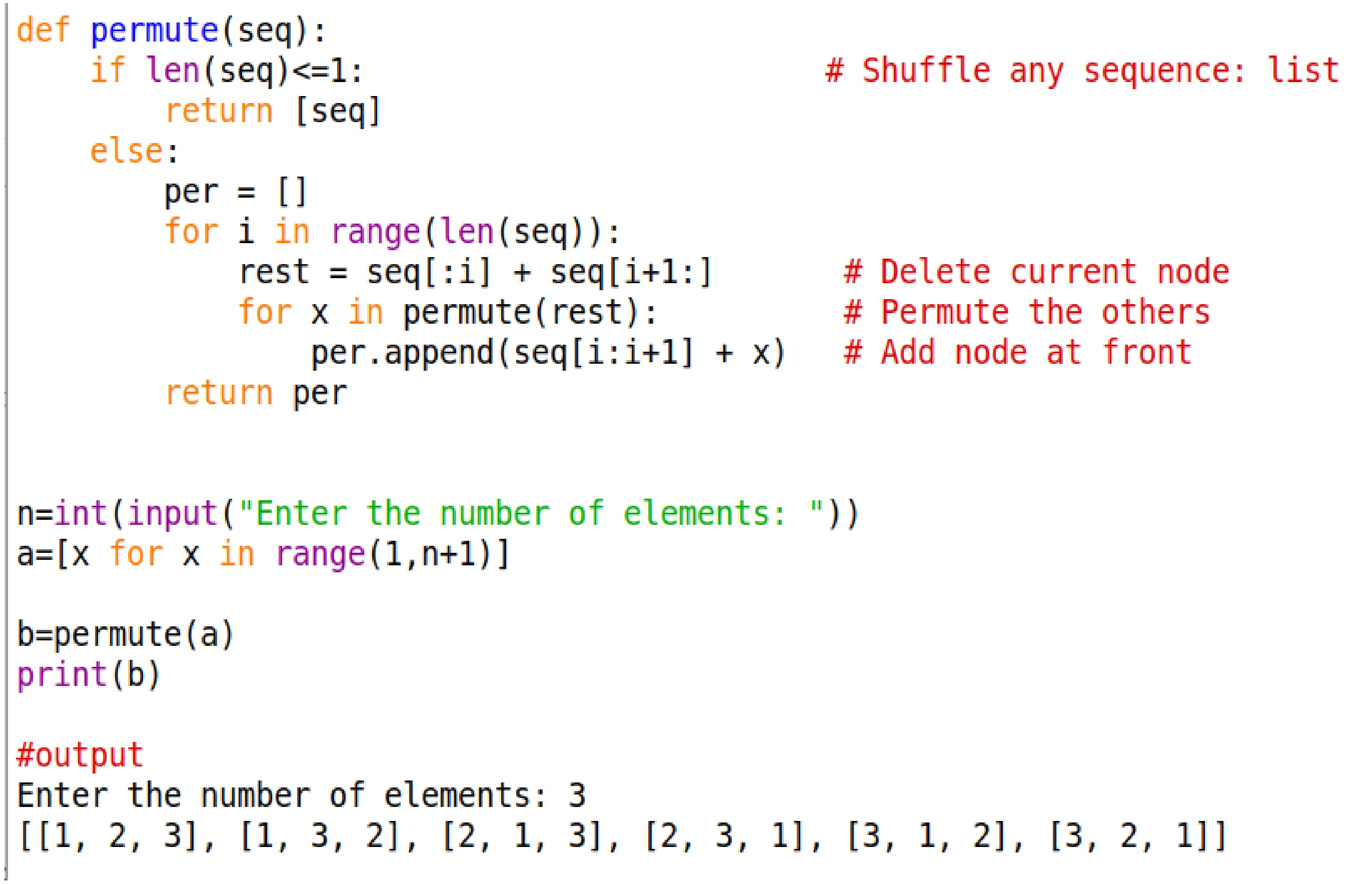


```
import math

def is_prime(number):
    if number > 1:
        if number == 2:
            return True
        if number % 2 == 0:
            return False
        for current in range(3, int(math.sqrt(number) + 1), 2):
            if number % current == 0:
                return False
        return True
    return False

def get_primes(number):
    while True:
        if is prime(number):
            number =yield number
        number += 1
```

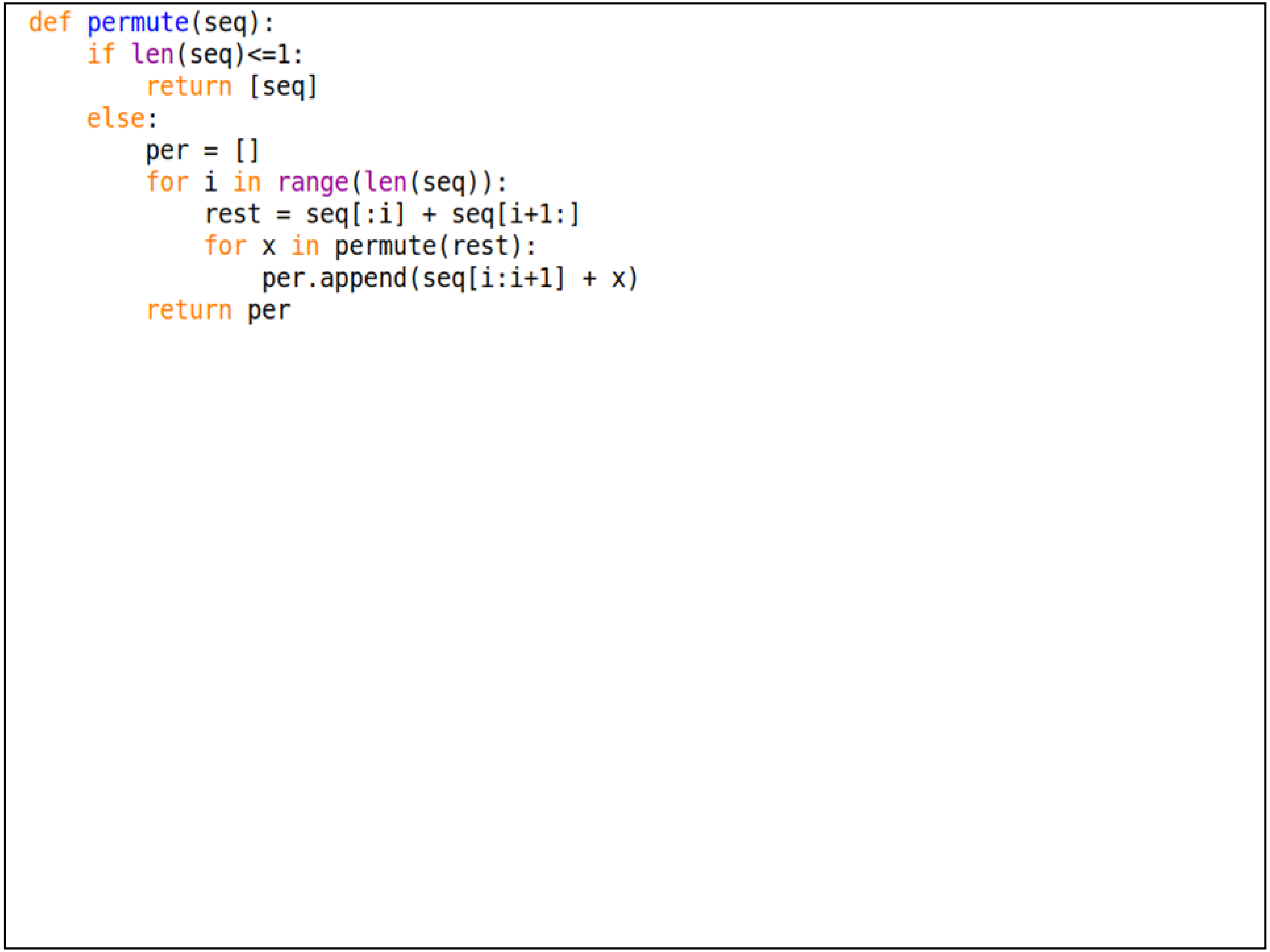


```
def get_primes(number):
    while True:
        if is prime(number):
            number = yield number
        number += 1


def print_successive_primes(iterations, base=2):
    prime_generator = get_primes(base)
    prime_generator.send(None)
    for power in range(iterations):
        print(prime_generator.send(base ** power))

num=int(input('Enter the number of iteration: '))
base=int(input('Enter the base: '))
print_successive_primes(num,base)

#output
Enter the number of iteration: 10
Enter the base: 10
2
11
101
1009
10007
100003
1000003
10000019
100000007
1000000007
```

Permutation



```
def permute(seq):
    if len(seq)<=1:                              # Shuffle any sequence: list
        return [seq]
    else:
        per = []
        for i in range(len(seq)):
            rest = seq[:i] + seq[i+1:]            # Delete current node
            for x in permute(rest):               # Permute the others
                per.append(seq[i:i+1] + x)        # Add node at front
        return per


n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

b=permute(a)
print(b)

#output
Enter the number of elements: 3
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```



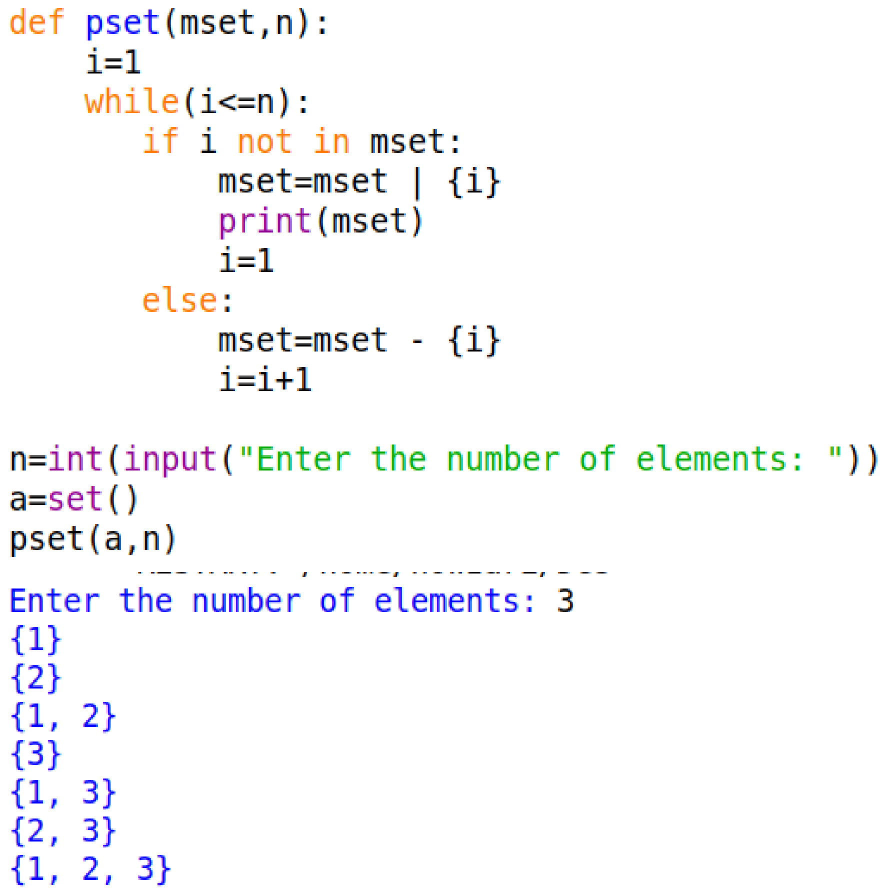
```python
def permute(seq):
    if len(seq)<=1:
        return [seq]
    else:
        per = []
        for i in range(len(seq)):
            rest = seq[:i] + seq[i+1:]
            for x in permute(rest):
                per.append(seq[i:i+1] + x)
        return per
```

Permutation

```
def permute(seq):
    if len(seq)<=1:                           # Shuffle any sequence
        yield seq
    else:
        for i in range(len(seq)):
            rest = seq[:i] + seq[i+1:]        # Delete current node
            for x in permute(rest):          # Permute the others
                yield seq[i:i+1] + x          # Add node at front

n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

b=permute(a)
print(list(b))

#output
Enter the number of elements: 3
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

```
def permute(seq):
    if len(seq)<=1:
        yield seq
    else:
        for i in range(len(seq)):
            rest = seq[:i] + seq[i+1:]
            for x in permute(rest):
                yield seq[i:i+1] + x
```

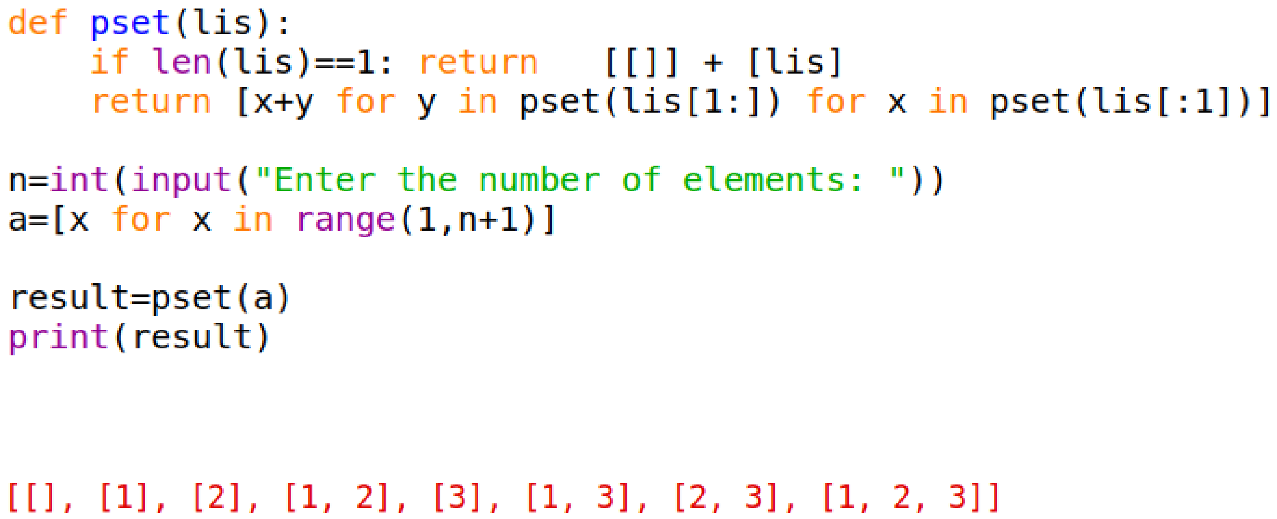
Power set



```
def pset(mset,n):
    i=1
    while(i<=n):
        if i not in mset:
            mset=mset | {i}
            print(mset)
            i=1
        else:
            mset=mset - {i}
            i=i+1

n=int(input("Enter the number of elements: "))
a=set()
pset(a,n)
```

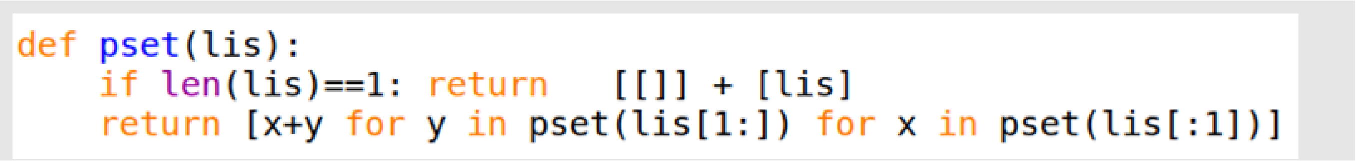
```
Enter the number of elements: 3
{1}
{2}
{1, 2}
{3}
{1, 3}
{2, 3}
{1, 2, 3}
```

Power set



```
def pset(lis):
    if len(lis)==1: return   [[]] + [lis]
    return [x+y for y in pset(lis[1:]) for x in pset(lis[:1])]

n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

result=pset(a)
print(result)
```

```
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
```

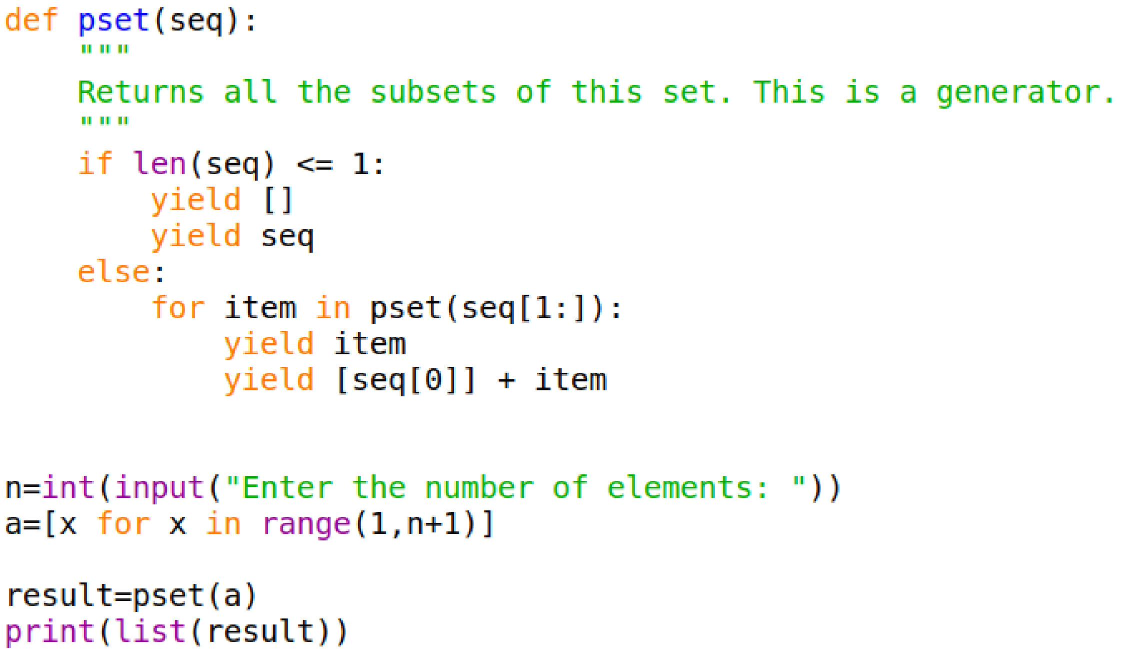


```
def pset(lis):
    if len(lis)==1: return   [[]] + [lis]
    return [x+y for y in pset(lis[1:]) for x in pset(lis[:1])]
```

Power set

```
def pset(lis):
    if len(lis)==1: return   [[]] + [lis]
    return (x+y for y in pset(lis[1:]) for x in pset(lis[:1]))

n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

result=pset(a)
print(list(result))
print()
```

Power set



```python
def pset(seq):
    """
    Returns all the subsets of this set. This is a generator.
    """
    if len(seq) <= 1:
        yield []
        yield seq
    else:
        for item in pset(seq[1:]):
            yield item
            yield [seq[0]] + item


n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

result=pset(a)
print(list(result))
```

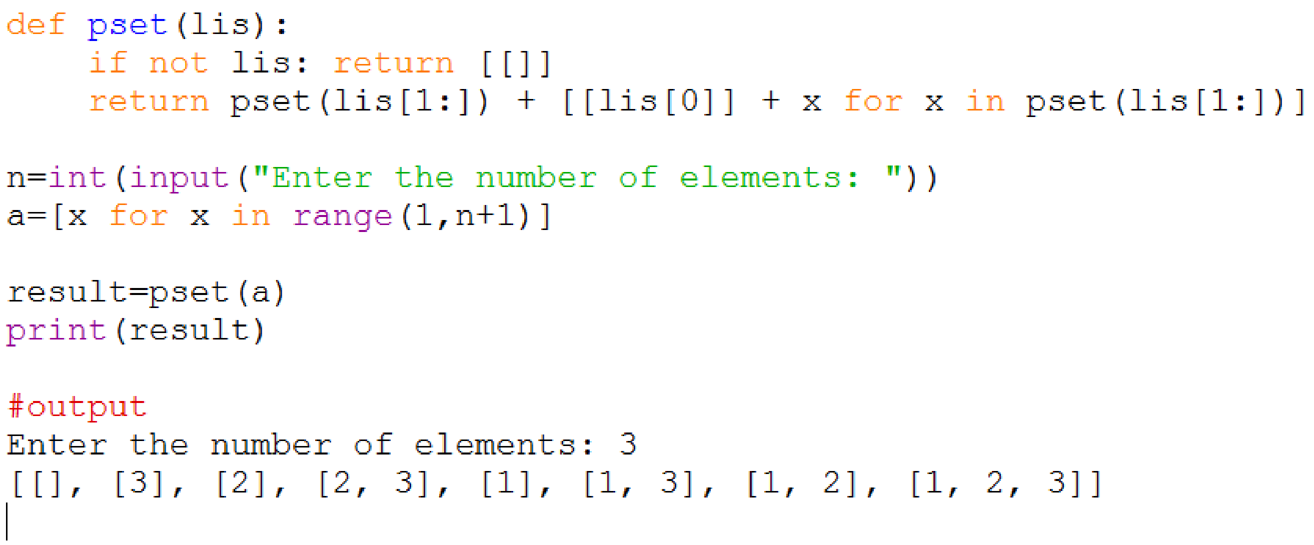
```
#output
Enter the number of elements: 3
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
```

```
def pset(seq):
 if len(seq) <= 1:
     yield []
     yield seq
 else:
     for item in pset(seq[1:]):
         yield item
         yield [seq[0]] + item
```

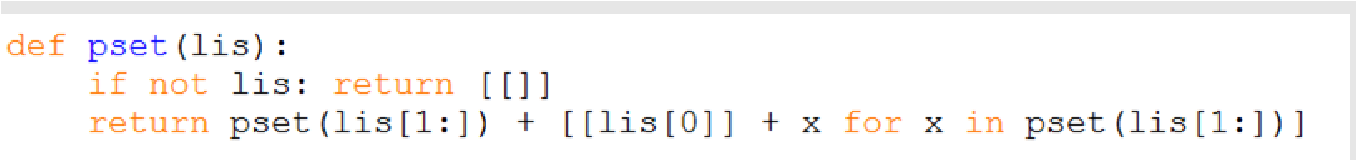
Power set : another order

```
def pset(seq):
    """
    Returns all the subsets of this set. This is a generator.
    """
    if len(seq) <= 1:
        yield seq
        yield []
    else:
        for item in pset(seq[1:]):
            yield [seq[0]]+item
            yield item

n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

result=pset(a)
print(list(result))

#output
Enter the number of elements: 3
[[1, 2, 3], [2, 3], [1, 3], [3], [1, 2], [2], [1], []]
```

Power set

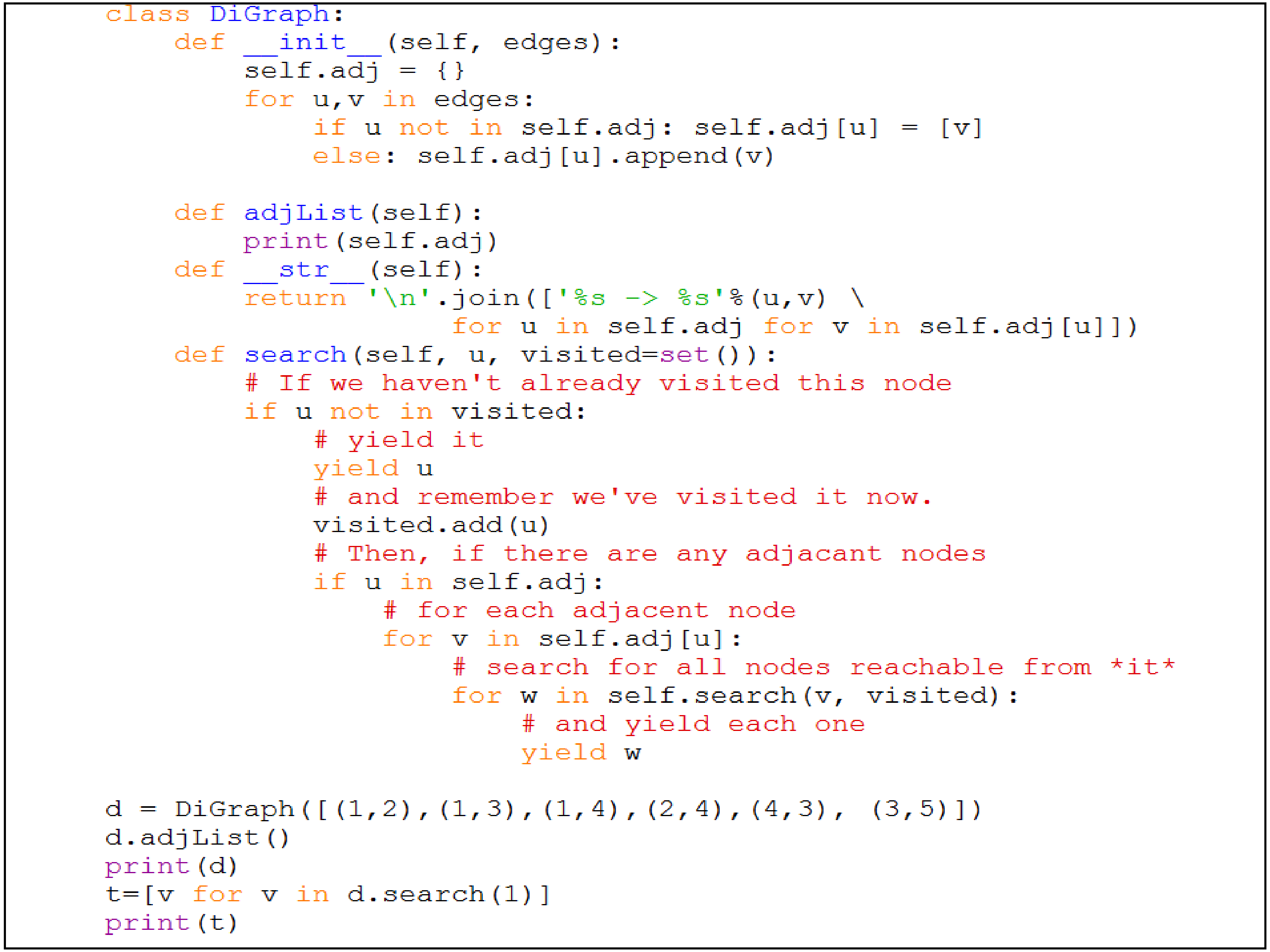


```
def pset(lis):
    if not lis: return [[]]
    return pset(lis[1:]) + [[lis[0]] + x for x in pset(lis[1:])]

n=int(input("Enter the number of elements: "))
a=[x for x in range(1,n+1)]

result=pset(a)
print(result)

#output
Enter the number of elements: 3
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
```

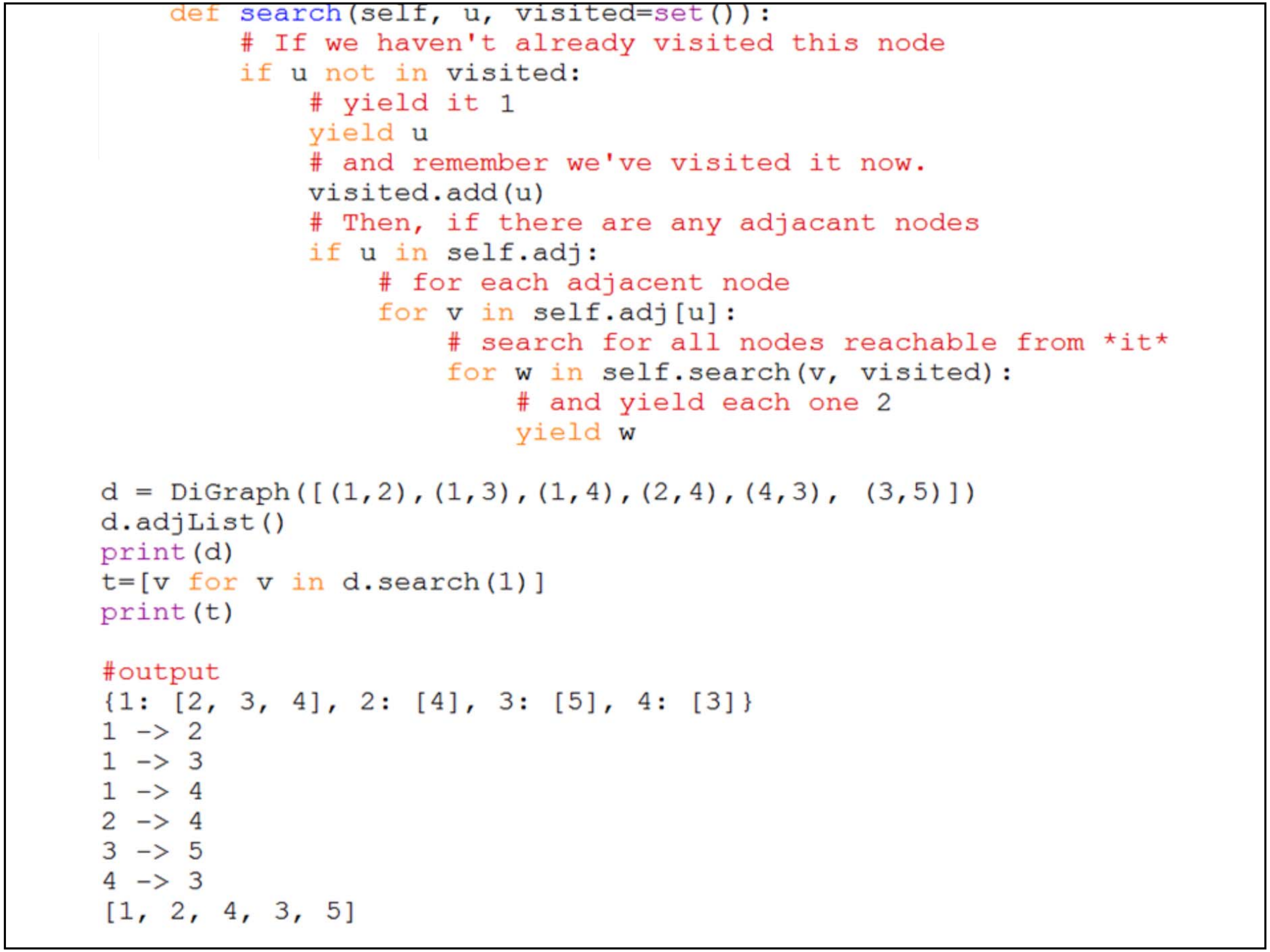


```
def pset(lis):
    if not lis: return [[]]
    return pset(lis[1:]) + [[lis[0]] + x for x in pset(lis[1:])]
```

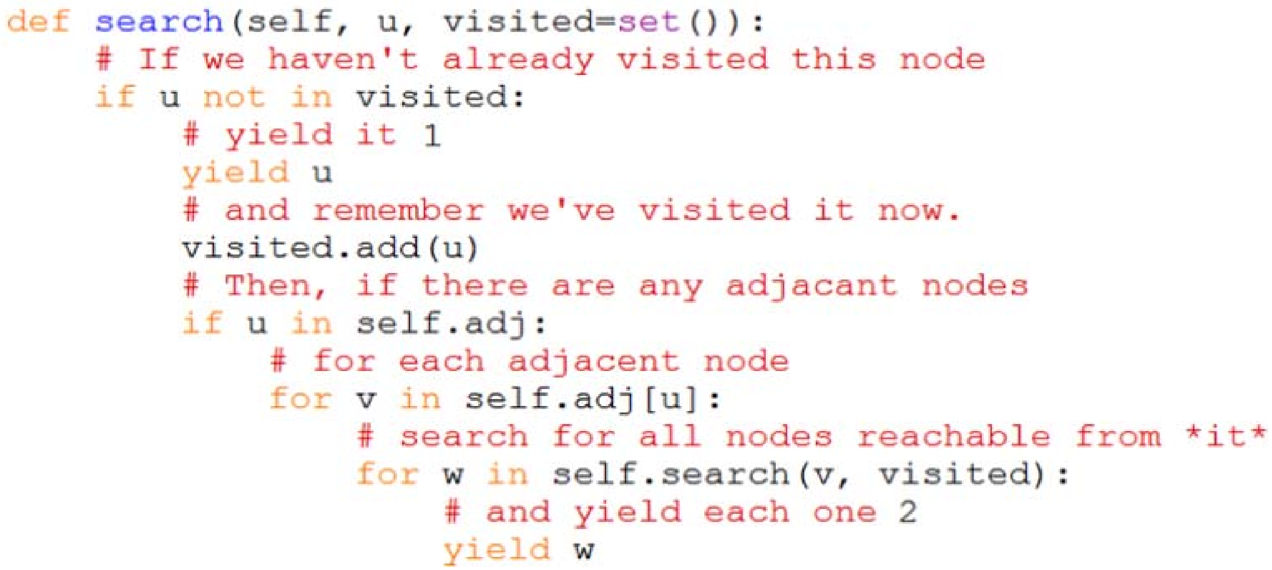
Directed Graph Search



```
class DiGraph:
    def __init__(self, edges):
        self.adj = {}
        for u,v in edges:
            if u not in self.adj: self.adj[u] = [v]
            else: self.adj[u].append(v)

    def adjList(self):
        print(self.adj)
    def __str__(self):
        return '\n'.join(['%s -> %s'%(u,v) \
                    for u in self.adj for v in self.adj[u]])
    def search(self, u, visited=set()):
        # If we haven't already visited this node
        if u not in visited:
            # yield it
            yield u
            # and remember we've visited it now.
            visited.add(u)
            # Then, if there are any adjacant nodes
            if u in self.adj:
                # for each adjacent node
                for v in self.adj[u]:
                    # search for all nodes reachable from *it*
                    for w in self.search(v, visited):
                        # and yield each one
                        yield w

d = DiGraph([(1,2),(1,3),(1,4),(2,4),(4,3), (3,5)])
d.adjList()
print(d)
t=[v for v in d.search(1)]
print(t)
```

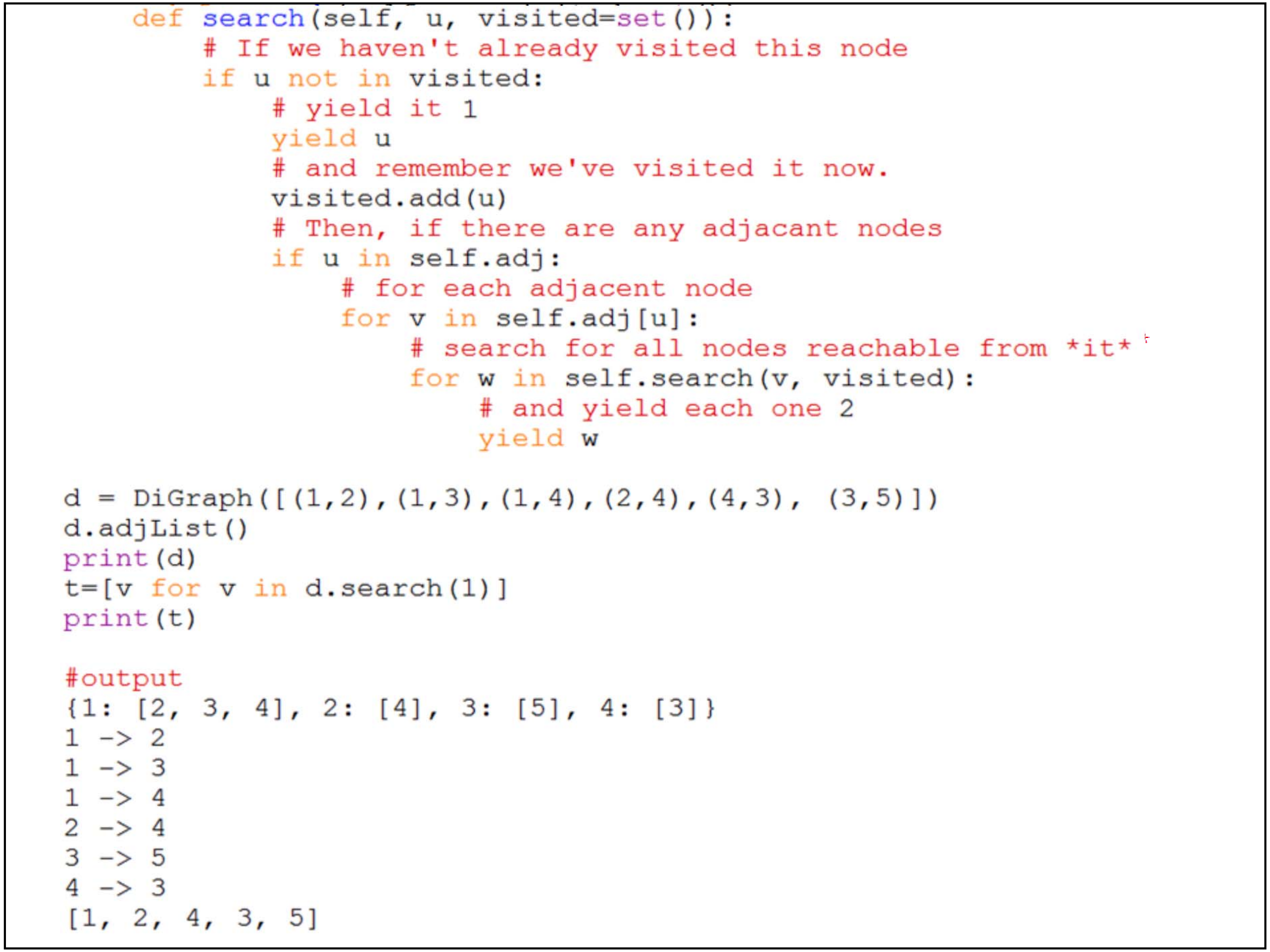


```
    def search(self, u, visited=set()):
        # If we haven't already visited this node
        if u not in visited:
            # yield it 1
            yield u
            # and remember we've visited it now.
            visited.add(u)
            # Then, if there are any adjacant nodes
            if u in self.adj:
                # for each adjacent node
                for v in self.adj[u]:
                    # search for all nodes reachable from *it*
                    for w in self.search(v, visited):
                        # and yield each one 2
                        yield w

d = DiGraph([(1,2),(1,3),(1,4),(2,4),(4,3), (3,5)])
d.adjList()
print(d)
t=[v for v in d.search(1)]
print(t)

#output
{1: [2, 3, 4], 2: [4], 3: [5], 4: [3]}
1 -> 2
1 -> 3
1 -> 4
2 -> 4
3 -> 5
4 -> 3
[1, 2, 4, 3, 5]
```



```python
def search(self, u, visited=set()):
    # If we haven't already visited this node
    if u not in visited:
        # yield it 1
        yield u
        # and remember we've visited it now.
        visited.add(u)
        # Then, if there are any adjacant nodes
        if u in self.adj:
            # for each adjacent node
            for v in self.adj[u]:
                # search for all nodes reachable from *it*
                for w in self.search(v, visited):
                    # and yield each one 2
                    yield w
```

End



```
    def search(self, u, visited=set()):
        # If we haven't already visited this node
        if u not in visited:
            # yield it 1
            yield u
            # and remember we've visited it now.
            visited.add(u)
            # Then, if there are any adjacant nodes
            if u in self.adj:
                # for each adjacent node
                for v in self.adj[u]:
                    # search for all nodes reachable from *it*
                    for w in self.search(v, visited):
                        # and yield each one 2
                        yield w

d = DiGraph([(1,2),(1,3),(1,4),(2,4),(4,3), (3,5)])
d.adjList()
print(d)
t=[v for v in d.search(1)]
print(t)

#output
{1: [2, 3, 4], 2: [4], 3: [5], 4: [3]}
1 -> 2
1 -> 3
1 -> 4
2 -> 4
3 -> 5
4 -> 3
[1, 2, 4, 3, 5]
```