This is to solve the Sage portion of Chapter 2 of

"A journey through the realm of numbers:

from quadratic equations to quadratic reciprocity."

I. Tuples, Lists and Sets

In [1]:
```
vector = (1, 2, 3)      # creates a tuple
vector[0]               # reads out the first entry
```

Out[1]:
```
1
```

In [2]:
```
numberlist = [1, 2, 3] # creates a list
numberlist[0]          # reads out the first entry
```

Out[2]:
```
1
```

In [3]:
```
numberlist[1] = 17     # overwrites the second entry
numberlist
```

Out[3]:
```
[1, 17, 3]
```

In [4]:
```
vector[1] = 17         # creates an error message
```

```
---------------------------------------------------------------------
---------
TypeError                                 Traceback (most recent c
all last)
<ipython-input-4-40d56c23b934> in <module>()
----> 1 vector[Integer(1)] = Integer(17)         # creates an erro
r message

TypeError: 'tuple' object does not support item assignment
```

In [5]:
```
x, y, z = vector # reads out all three entries and assigns the
                 # three variables with its values
print(x, y, z)
```

```
1 2 3
```

In [6]:
```
print(range(11))
print(range(1,11))
print([n for n in range(1,11,2)])
print(srange(0,1,1/10))
```

```
range(0, 11)
range(1, 11)
[1, 3, 5, 7, 9]
[0, 1/10, 1/5, 3/10, 2/5, 1/2, 3/5, 7/10, 4/5, 9/10]
```

In [7]:
```
[p for p in range(50) if is_prime(p)] # 'if' after 'for' selects
                                      # some entries
```

Out[7]:
```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

In [8]:
```
[[10*m+n for m in range(3)] for n in range(3)] # list of lists
```

Out[8]:
```
[[0, 10, 20], [1, 11, 21], [2, 12, 22]]
```

In [9]:
```
[10*m+n for m in range(3) for n in range(3)]   # iterated for, but
                                               # creating only one
                                               # list of numbers
```

Out[9]:
```
[0, 1, 2, 10, 11, 12, 20, 21, 22]
```

In [10]:
```
tuple(srange(0,1,1/3))
```

Out[10]:
```
(0, 1/3, 2/3)
```

In [11]:
```
Set("Which characters are contained in this sentence?")
```

Out[11]:
```
{'W', 't', 'e', 'n', 'i', '?', 'd', 'h', ' ', 'a', 'r', 's', 'c',
'o'}
```

In [12]:
```
Set([1,2,2,3,2,1,3])
```

Out[12]:
```
{1, 2, 3}
```

In [13]:
```
"x" in "Which characters are contained in this sentence?"
```

Out[13]:
```
False
```

In [14]:
```
2 in range(5)
```

Out[14]:
```
True
```

In [15]:
```
for k, char in enumerate("index"): # for strings, lists, tuples
    print(str(k)+"th character in 'index' is "+char)
```

```
0th character in 'index' is i
1th character in 'index' is n
2th character in 'index' is d
3th character in 'index' is e
4th character in 'index' is x
```

II. Projects

In [16]:
```
def mediant(r,s):
    return ( (r.numerator()+s.numerator())
             / (r.denominator()+s.denominator()) )

mediant(0,1)
```

Out[16]:
```
1/2
```

In [17]:
```
def Farey(n):
    if n==1:
        return [0,1]
    else:
        F_old=Farey(n-1)
        F_mediant=[mediant(F_old[k],F_old[k+1])
                           for k in range(len(F_old)-1)]
        F_new=[r for r in F_mediant if r.denominator()==n]
        return sorted(F_old+F_new)
```

In [18]:
```
Farey(5)
```

Out[18]:
```
[0, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1]
```

In [19]:
```
A = Set("Sage")
B = Set(range(3))
Set([(a,b) for a in A for b in B]) # a single line gives the
                                   # Cartesian product
```

Out[19]:
```
{('S', 0), ('a', 0), ('S', 2), ('a', 1), ('a', 2), ('g', 2), ('g',
1), ('e', 0), ('g', 0), ('e', 1), ('e', 2), ('S', 1)}
```

In [20]:
```
C = Set("Domain of function")
def f(c):
    return c.upper()        # gives the upper case letter of c

Set([f(c) for c in C])      # a single line gives the image f(C)
```

Out[20]:
```
{'D', 'U', 'I', 'F', 'N', 'T', 'A', ' ', 'C', 'M', 'O'}
```

In [21]:
```
all(a>0 for a in range(3))  # recall that range(3)=[0,1,2]
```

Out[21]: False

In [22]:
```
any(a>0 for a in range(3))
```

Out[22]: True

In [23]:
```
def maps_into(f, A, B):
    return all(f(a) in B for a in A)
```

In [24]:
```
maps_into(f, Set("Sage"), Set("Math"))
```

Out[24]: False

In [25]:
```
maps_into(f, Set("Sage"), Set("SAGEMATH"))
```

Out[25]: True

In [26]:
```
def injective_on(f, A):
    return all((a1==a2 or f(a1)!=f(a2)) for a1 in A for a2 in A)
```

In [27]:
```
injective_on(f,Set("Sage"))
```

Out[27]: True

In [28]:
```
injective_on(f,Set("Sage and sage")) # "S" and "s" both map to "S"
```

Out[28]: False

In [29]:
```
def surjective_to(f,A,B):
    Image = Set([f(a) for a in A])    # store Image to avoid it
    return all(b in Image for b in B) # being calculated often
```

In [30]:
```
surjective_to(f,Set("Sage"),Set("S"))
```

Out[30]: True

In [31]:
```
surjective_to(f,Set("Sage"),Set("Sage"))
```

Out[31]: False

In [32]:
```
def bijective_between(f, A, B):
    return (maps_into(f, A, B) and
            injective_on(f, A) and
            surjective_to(f, A, B))
```

In [33]:
```
bijective_between(f, Set("Sage"), Set("SAGE"))
```

Out[33]: True

In [34]:
```
bijective_between(f, Set("Sage"), Set("SAGEMATH"))
```

Out[34]:
```
False
```

In [35]:
```
def inverse_applied_to(f, A, b):
    return Set([a for a in A if f(a)==b])
```

In [36]:
```
inverse_applied_to(f, Set("SageMATH"), "A")
```

Out[36]:
```
{'A', 'a'}
```

In [37]:
```
def powerset(A):
    if A.is_empty():
        return Set([Set([])])
    else:
        a = A.an_element()
        P_old = powerset(A.difference(a))
        return P_old + Set([S + Set([a]) for S in P_old])
```

In [38]:
```
powerset(Set("abc"))
```

Out[38]:
```
{{'c', 'a', 'b'}, {'c', 'a'}, {'b'}, {'c', 'b'}, {}, {'a', 'b'}, {
'a'}, {'c'}}
```

In [39]:
```
[S for S in powerset(Set("abc")) if S.cardinality()==1]
```

Out[39]:
```
[{'b'}, {'a'}, {'c'}]
```

In [40]:
```
def powerset_ordered(A):
    P = powerset(A)
    n = len(A)
    P_ordered=[]
    for k in range(n+1):
        P_ordered = P_ordered\
                    +[S for S in P if S.cardinality()==k]
    return P_ordered
```

In [41]:
```
powerset_ordered(Set("abc"))
```

Out[41]:
```
[{}, {'b'}, {'a'}, {'c'}, {'c', 'a'}, {'c', 'b'}, {'a', 'b'}, {'c'
, 'a', 'b'}]
```

In [42]:
```
def ordinal(n):
    if n == 0:
        return Set([])
    else:
        lastone = ordinal(n-1)
        return(lastone + Set([lastone]))
```

In [43]:
```
for n in range(5):
    print(ordinal(n), "has", ordinal(n).cardinality(), "elements")
```

```
{} has 0 elements
{{}} has 1 elements
{{}, {{}}} has 2 elements
{{}, {{}}, {{}, {{}}}} has 3 elements
{{}, {{}, {{}}, {{}, {{}}}}, {{}}, {{}, {{}}}} has 4 elements
```

In [ ]: