The set of all Python strings of ASCII codes is “no more numerous” than the set of natural numbers.

Proof:

Let **C** be the set of every Python ASCII codes. So,

C = {'\x00', '\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t', '\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14', '\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e', '\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f'}

Here, cardinality of **C**, $|\mathbf{C}| = 128$

The set **S** is the set of every possible Python strings obtained by:

$$\mathbf{C}^0 \cup \mathbf{C}^1 \cup \mathbf{C}^2 \cup \mathbf{C}^3 \cup \ldots\ldots \cup \mathbf{C}^{\mathbb{N}}$$

So,

$$\mathbf{S} = \bigcup_{i=0}^{\mathbb{N}} \mathbf{C}^i$$

No two natural numbers greater than 1 have the same prime factorization $p_1^{a1} * p_2^{a2} * p_3^{a3} * \ldots\ldots * p_n^{an}$, where ‘$p_n$’ is a prime number and ‘an’ is a natural number; or equivalently, every natural number greater than 1 can be written as a product of prime numbers, and this product is unique. **[**from the Fundamental Theorem of Arithmetic**]** ………**(Statement 1)**

The following Python program has two functions. The function **f(x)** takes a Python string **x** as input and returns a natural number. The function **generate_n_primes(N)** is a simple helper function for the main function ‘f(x)’ and it returns a Python list of first N prime numbers.

```
def generate_n_primes(N):
  primes  = []
  chkthis = 2
  while len(primes) < N:
    ptest    = [chkthis for i in primes if chkthis%i == 0]
    primes  += [] if ptest else [chkthis]
    chkthis += 1
  return primes

def f(x):
  tpString = tuple(list(x))
  lenn = len(tpString)
  lsPrimes = generate_n_primes(lenn)
  prod = 1
  for i in range(lenn):
    prod*=lsPrimes[i]**ord(tpString[i])
  return prod
```

Here,

tpString is a tuple formed by splitting the string **x** into its constituent Python characters

So, for a string x = “def f(x): return (x)”

tpString = ('d', 'e', 'f', ' ', 'f', '(', 'x', ')', ':', ' ', 'r', 'e', 't', 'u', 'r', 'n', ' ', '(', 'x', ')')

lenn stores the length of the tuple ‘tpString’. For the example above, lenn = 20

In the program, ‘lenn’ is passed as the argument for ‘N’. So, for the example above, the variable **lsPrimes** stores the list of first 20 prime numbers returned by the function call ‘generate_n_primes(lenn)’

The in-built Python function **ord(c)** returns the integer representing the Unicode code-point of the passed Python character ‘c’, which is unique for every character ………**(Statement 2)**

After the execution of the following code portion in the Python program above,

```
prod = 1
for i in range(lenn):
    prod*=lsPrimes[i]**ord(tpString[i])
```

the variable **prod** will store a natural number whose product form looks like:

$p_1^{a1} * p_2^{a2} * p_3^{a3} * \ldots\ldots * p_i^{ai}$

where ‘p_i‘ corresponds to the ‘i’th prime number in the list ‘lsPrimes’ and ‘ai’ corresponds to the value returned by the function call ‘ord(tpString[i])’, which is a unique Unicode code-point integer value of the ‘i’th character in the tuple ‘tpString’, and the function ‘f(x)’ returns the natural number stored in ‘prod’ …….**(Statement 3)**

Let, a number 'n' belong to the set of natural numbers. So,

For some set $\mathbf{C}^n$,

Let every element be represented by an n-tuple (t1, t2, t3,…tn)

No two elements will be equal **[**properties of sets**]**

Therefore,

the tuple (ord(t1), ord(t2), ord(t3), …ord(tn)) will be unique to every element, where the function ord(tn) is the same in-built Python function used in the program above **[**from statement 3**]** **…………………………………………………………………………………….(Statement 4)**

Therefore, the tuple (a1, a2, a3,…an) will be unique to every element, where an = ord(tn) **[**from statement 4**]..(Statement 5)**

Therefore, for no two elements (a1, a2, a3,…an) and (b1, b2, b3,…bn), ak = bk {where 'k' ranges from 1 to n} **[**from statement 5**]** ……**(Statement 6)**

We know,

every product $p_1^{a1} * p_2^{a2} * p_3^{a3} * . . . . * p_i^{ai}$ is mapped to only one natural number {where, $p_1^{a1} * p_2^{a2} * p_3^{a3} * . . . . . * p_n^{an}$, where '$p_i$' is a prime number and 'ai' is a natural number} **[**from statement 1**]** **…………………………………………………………………………………….(Statement 7)**

Let, tp1 = (m1, m2, m3,…mi) and tp2 = (n1, n2, n3,…nj) be two tuples, where 'mk' and 'nk' are integers where k ranges in integers {1..i} and {1..j} for tp1 and tp2 respectively

Two products $p_1^{m1} * p_2^{m2} * p_3^{m3} * . . . . * p_i^{mi}$ and $p_1^{n1} * p_2^{n2} * p_3^{n3} * . . . . * p_j^{nj}$ corresponding to the two strings 'xm' and 'xn' respectively passed to the function 'f(x)' in statement 3 will be equal if and only if the tuples

tp1 = tp2 **[**from statements 3 and 7**]** **……….(Statement 8)**

The function 'f(x)' in statement 3 will return a same natural number for strings 'xm' and 'xn' if and only if tuples tp1 = tp2 in statement 8 [from statements 3 and 8**]** **…….(Statement 9)**

Tuples tp1 = tp2, if and only if:

i = j = k (constant), and

for every integer ‘z’ in {1..k}, mz = nz **[**properties of tuples**]** **…………………………………………………………………….……………….(Statement 10)**

But we know, even for i = j = k, and for every ‘z’ in {1..k},

mz is not equal to nz **[**from statement 6**]** **……………………………………………………………………………………..……………………….(Statement 11)**

Therefore, tp1 is not equal to tp2 **[**from statements 10 and 11**]** **…………………………..………………………………………………………..…….(Statement 12)**

Therefore, for any two Python strings ‘xm’ and ‘xn’, the function ‘f(x)’ will return two unique natural numbers **[**from statements 9 and 12**]** **……….(Statement 13)**

Therefore, the function ‘f(x)’ is an injection from the set ‘S’ to the set of natural numbers, which proves that **the set of all Python strings of ASCII codes is “no more numerous” than the set of natural numbers.** **[**from statement 13**]**

-By Pradip Lamsal