Numeral Base Evaluator

Bin Han

Project Introduction

A number like 235 in decimal is a notation for the following expression:

$2 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10 = 200 + 30 + 5$
$= 235$
We can take any other integer as the base of our number system. In base 9 (nonary), 235 would be a notation for the following expression:

$2 \cdot 9^2 + 3 \cdot 9^1 + 5 \cdot 9 = 162 + 27 + 5$
which evaluates to 194 in decimal. In the course, we have learned the representation of numbers in binary, i.e. base 2.

Write a function which can take a notation in any base and compute its value. A base will be defined by a dictionary which maps digits (as characters) to their value (as integers).

```
binary = {'0': 0, '1': 1}
decimal = {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
'8': 8, '9': 9}
nonary = {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
'8': 8}
hexadecimal = {
    '0': 0,  '1': 1,  '2': 2,  '3': 3,  '4': 4,
    '5': 5,  '6': 6,  '7': 7,  '8': 8,  '9': 9,
    'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15
}
```

1. define a function parse(base, notation) which takes notation as a string, and returns a list of integers l, so that l[i] is the value of the digit found in position i in the string, as an integer. If any digit is not valid, then return None instead of a list. The parameter base is a base, as defined above.

```
def parse(base, notation):
   list=[]
   for i in range(len(notation)):
     if notation[i] in base:
         list.append(base[notation[i]])
     else:
        return None
        break
   return list
print(parse(binary, "235"))
print(parse(binary, "1101"))
```

```
None
[1, 1, 0, 1]
```

2. define a function powers(m,n), where m is the number of digits in the base as an integer, and returns a list of powers of m of length n.

```
def powers(m,n):
  list=[]
  for i in range(n):
    list.append(m**i)
  list.reverse()
  return list
print(powers(2, 5))
print(powers(3, 6))
print(powers(4, 8))
```

```
[16, 8, 4, 2, 1]
[243, 81, 27, 9, 3, 1]
[16384, 4096, 1024, 256, 64, 16, 4, 1]
```

3. Define a function n_ary(base, notation) that takes the base, and the notation as a string. Return None if the notation is invalid for the given base, or its value as an integer otherwise. You must call the above functions to structure the function.

```
def n_ary(base,notation):
  if parse(base,notation) is not None:
    sum=0
    n=len(notation)
    for i in range(n):
```

```
        sum=sum+int(base[notation[n-1-i]])*(len(base)**i)
    return sum
  else:
    return None
print(n_ary(decimal, "235"))
print(n_ary(nonary, "235"))
print(n_ary(binary, "235"))
print(n_ary(binary, "1011"))
print(n_ary(hexadecimal, "FF"))
```

```
235
194
None
11
255
```

4. Define a function make_base(n) which creates the usual base for any n less or equal to 10. For example, base(2) equals binary as binary.

```
def make_base(n):
  dict={}
  for i in range(n):
     dict[str(i)]=i
  return dict
print(make_base(2))
print(make_base(3))
print(make_base(4))
```

```
{'0': 0, '1': 1}
{'0': 0, '1': 1, '2': 2}
{'0': 0, '1': 1, '2': 2, '3': 3}
```