Algorithm for checking prime number

```python
import sys

def isPrime(n):
    for i in range(2, n/2 + 1, 1):
        if n % i == 0:
            return False
    else:
        return True

n = int(sys.argv[1])
output = '{0} : {1}'.format(n, isPrime(n))

print output
```

Output

```
$  python isPrime.py 577
```

- 577 : True

Algorithm for finding factors

```
import  sys

def findFactors(n):
    factors = []
    for i in range(1, n//2 + 1, 1):
        if n % i == 0:
            factors.append(i)
    else:
        factors.append(n)
        return factors

n = int(sys.argv[1])
output = '{0} : {1}'.format(n, findFactors(n))

print output
```

Output

```
$  python findFactors.py 24
```

- `24 :  [1, 2, 3, 4, 6, 8, 12, 24]`

recursive algorithm for factorial

```python
import sys

def factorial(n):
        if n == 0:
            return 1
        else:
            return n * factorial(n-1)

n = int(sys.argv[1])
output = '{0}! = {1}'.format(n, factorial(n))

print(output)
```

Output

```
$  python factorial.py 5
```

- `5!  = 120`

modeling polynomial - only addition

```
class Polynomial:
    def __init__(self, *coeffs):
        self.coeffs = coeffs

    def __repr__(self):
        return 'Polynomial{}'.format(self.coeffs)

    def __add__(self, other):
        return Polynomial(*(x+y for x, y in
         ↪  zip(self.coeffs, other.coeffs)))
```

Output

```
$ python -i polynomial.py
> p = Polynomial(3, 5, 2) # 3x^2+5x+2
> q = Polynomial(7, 2, 3) # 7x^2+2x+3
> p + q
• Polynomial(10, 7, 5) # 10x^2+7x+5
```


modeling of polynomial - multiplication

```
class Polynomial:
    def __init__(self, *coeffs):
        self.coeffs = coeffs

    def __repr__(self):
        return 'Polynomial{}'.format(self.coeffs)

    def deg(self):
        return len(self.coeffs)-1

    def __mul__(self, other):
        pol = [0]*(self.deg()+other.deg()+1)

        for x in range(len(self.coeffs)):
            for y in range(len(other.coeffs)):
                pol[x+y] += self.coeffs[x] *
                 ↪  other.coeffs[y]

        return Polynomial(*(c for c in pol))
```