Python Recap – I

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 1

Computing gcd

- $\text{gcd}(m, n)$ — greatest common divisor
    - Largest k that divides both m and n
    - $\text{gcd}(8, 12) = 4$
    - $\text{gcd}(18, 25) = 1$
    - Also hcf — highest common factor

Computing gcd

- $\mathrm{gcd}(m,n)$ — greatest common divisor
  - Largest k that divides both m and n
  - $\mathrm{gcd}(8,12) = 4$
  - $\mathrm{gcd}(18,25) = 1$
  - Also hcf — highest common factor
- $\mathrm{gcd}(m,n)$ always exsits
  - 1 divides both m and n

Computing gcd

- $\mathrm{gcd}(m,n)$ — greatest common divisor
  - Largest k that divides both m and n
  - $\mathrm{gcd}(8,12) = 4$
  - $\mathrm{gcd}(18,25) = 1$
  - Also hcf — highest common factor
- $\mathrm{gcd}(m,n)$ always exsits
  - 1 divides both m and n
- Computing $\mathrm{gcd}(m,n)$
  - $\mathrm{gcd}(m,n) \leq \min(m,n)$
  - Compute list of common factors from 1 to $\min(m,n)$
  - Return the last such common factor

```python
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Points to note

- Need to initialize `cf` for `cf.append()` to work
  - Variables (names) derive their type from the value they hold

```
def gcd(m,n):
  cf = []    # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Points to note

- Need to initialize cf for cf.append() to work
  - Variables (names) derive their type from the value they hold
- Control flow
  - Conditionals (if)
  - Loops (for)

```
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Points to note

- Need to initialize `cf` for `cf.append()` to work
  - Variables (names) derive their type from the value they hold
- Control flow
  - Conditionals (`if`)
  - Loops (`for`)
- `range(i,j)` runs from `i` to `j-1`

```
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Points to note

- Need to initialize `cf` for `cf.append()` to work
  - Variables (names) derive their type from the value they hold
- Control flow
  - Conditionals (`if`)
  - Loops (`for`)
- `range(i,j)` runs from `i` to `j-1`
- List indices run from `0` to `len(l) - 1` and backwards from `-1` to `-len(l)`

```
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Eliminate the list

- Only the last value of cf is important

```
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Eliminate the list

- Only the last value of cf is important
- Keep track of most recent common factor (mrcf)

```
def gcd(m,n):
 for i in range(1,min(m,n)+1):
  if (m%i) == 0 and (n%i) == 0:
   mrcf = i
 return(mrcf)
```

Computing gcd

Eliminate the list

- Only the last value of `cf` is important
- Keep track of most recent common factor (`mrcf`)
- Recall that `1` is always a common factor
    - No need to initialize `mrcf`

```
def gcd(m,n):
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      mrcf = i
  return(mrcf)
```

Computing gcd

Eliminate the list

- Only the last value of cf is important
- Keep track of most recent common factor (mrcf)
- Recall that 1 is always a common factor
  - No need to initialize mrcf

Efficiency

- Both versions of gcd take time proportional to $\min(m, n)$

```
def gcd(m,n):
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      mrcf = i
  return(mrcf)
```

```
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```

Computing gcd

Eliminate the list

- Only the last value of `cf` is important
- Keep track of most recent common factor (`mrcf`)
- Recall that `1` is always a common factor
  - No need to initialize `mrcf`

Efficiency

- Both versions of `gcd` take time proportional to $\min(m, n)$
- Can we do better?

```
def gcd(m,n):
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      mrcf = i
  return(mrcf)
```

```
def gcd(m,n):
  cf = []   # List of common factors
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)
  return(cf[-1])
```