CptS 355- Programming Language Design

Python
Higher-Order Functions

Instructor: Sakire Arslan Ay

Fall 2023

WASHINGTON STATE UNIVERSITY

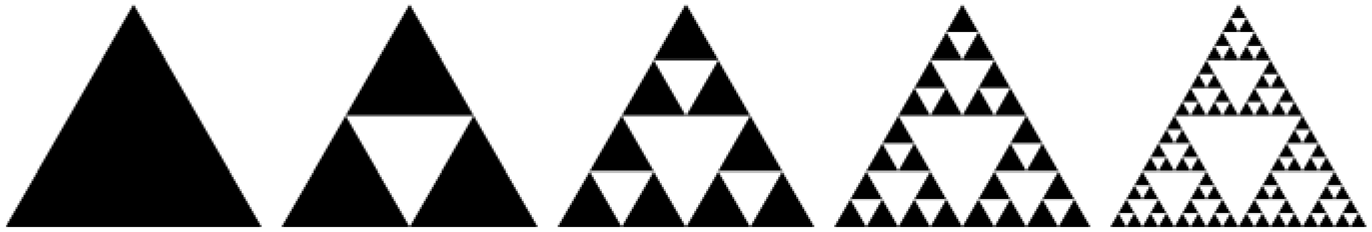
World Class. Face to Face.

Lecture material

- Please watch the Python part-1 and part2 videos on Canvas.
- No lecture notes on Python basics
- Lecture notes on:
  - Python lists
  - Python dictionaries
  - Higher order functions, recursion
  - Classes, iterators, streams

Recursive Functions

- A function is called recursive if the body of that function calls itself, either directly or indirectly.



Iteration vs Recursion

- Iteration is a special case of recursion
  - Example: factorial
    - 4! = 4*3*2*1

Iterative Control:

MATH:

$$n! = \prod_{i=1}^{n} i$$

Names:

```
n, total, k
```

Recursion:

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

```
n
```

Using iterative control:

```
def fact_iter(n):
    total, k = 1, 1
    while k <= n:
        total, k = total*k, k+1
    return total
```

Using recursion:

```
def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

Iteration vs Recursion

- Example: reverse

Recursion:

```
def reverse(s):
   if s == '':
      return s
   return reverse(s[1:]) + s[0]
```

Using iterative Control:

```
def reverse2(s):
    r = ''
    i = 0
    while i < len(s):
        r = s[i] + r
        i = i + 1
    return r
```

Higher Order Functions- map, reduce, filter

- map/transform
  - map takes a unary function and a list and produces a same-sized list of mapped/transformed values based on substituting each value with the result of calling the parameter function on it.
  - For example,

```
def sq(x):
   return x**2        #i.e.,x²
L = [i for i in range(0,5)]
map( sq, L )
```

Higher Order Functions- map, reduce, filter

- map/transform
  - Here is a simple implementation of the map function.

```
def map(f,alist):
    answer = []
    for v in alist:    # generate each value v in a list
       answer.append(f(v)) # put (v) in a list to return
    return answer
```

  - Python's built-in map function is more general and faster.

```
map ((lambda x,y: x+y),[1,2,3,4],[5,6,7,8])
```

returns ?

Higher Order Functions- map, reduce, filter

- filter
  - Filter takes a predicate (a unary function returning a bool) and some list of values and produces a list with only those values for which the predicate returns True (or a value that is interpreted as True).
  - For example:

```
filter((lambda x: x>0), [-4,3,1,-2,3,-5,1,9,0])
```

returns ?

Higher Order Functions- map, reduce, filter

- filter
  - Here is a simple implementation of the filter function.

```
def filter(p,alist):
    answer = []
    for v in alist:
       if p(v):
           answer.append(v)
    return answer
```

  - Python's built-in filter function is faster

Higher Order Functions- map, reduce, filter

- reduce (foldr/foldl or accumulate)
  – Reduce takes a binary function and some list of values and reduces or accumulates these results into a single value.
  – For example:

```
from functools import reduce
reduce((lambda x,y : x+y), [i for i in range(1,100)] )
reduce( max, [4,2,-3,8,6] )
```

- Unlike `map` and `filter` (which are defined and automatically imported from the builtins module) we must import reduce from the `functools` module explicitly.

Higher Order Functions- map, reduce, filter

- reduce
  - Here is a simple basic implementation of the reduce function.

```
def reduce(f,alist):
    if alist == []:
        return None

    answer = alist[0]
    for v in alist[1:]:
        answer = f(answer,v)
    return answer
```

Higher Order Functions- map, reduce, filter

- reduce
  - Here is a sample implementation of the reduce function.

```
def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value
```

Higher Order Functions- map, reduce, filter

Additional remarks:

- The map, filter, and reduce function work on anything that is iterable (which list and tuple are, but so are sets, dicts, and even strings).
  - We can call `map(lambda x : x.upper(), 'Hello')` to produce the list `['H','E','L','L','O']`.

- The map and filter functions built into Python produce an iterable as a result (not a real list). So if we call:

  `print(map(lambda x : x.upper(),'Hello'))`

  prints `<map object at 0x02DFFE30>`

  We need to create a list from that iteratable object, i.e.,

  `print(list(map(lambda x : x.upper(), 'Hello')))`

  Python prints: `['H','E','L','L','O']`