Programming Paradigms

style, mode of thinking in solving problems, abstraction over Von Neumann model
Imperative Programming

A series of commands that change the state of the machine to solve a problem.

1. Do A.
2. Do B.
3. Compare.
4. Branch.
5. Do D.
Object-Oriented Programming

Objects have data and methods; data takes “center stage” rather than processing, and relationships become important.
Functional Programming

Evaluating a collection of functions that have defined inputs and outputs.

transformation rather than modification

Make small changes over large groups of data, doing this several times to solve problem. Works well for lists, dictionaries, etc.
Functional Programming

Inspired by Alonzo Church’s lambda calculus — a formal way for writing and using functions.

three most common transformations:

Mapping

Filtering

Reduction
Mapping

one-to-one

example:  \(x^2 + 1\)

\([1, 2, 3, 4, 5]\) becomes \([3, 5, 7, 9, 11]\)
Filtering

- testing each value to retain or not

example: keep odd values

\[ \{1, 2, 3, 4, 5\} \text{ becomes } \{1, 3, 5\} \]
Reduction

applying a binary (two-value not 0/1) function cumulatively

eample: \[1, 2, 3, 4, 5\] would become \(((1+2)+3)+4)+5\) which is 15
lambda functions

```python
>>> def f(x): return x**2
...  
>>> print f(8)  
64  
>>>  
>>> g = lambda x: x**2  
>>>  
>>> print g(8)  
64  

>>> def make_incrementor(n): return lambda x: x + n

>>> f = make_incrementor(2)  
>>> g = make_incrementor(6)  
>>>  
>>> print f(42), g(42)  
44 48  
>>>  
>>> print make_incrementor(22)(33)  
55
```
One can use regular functions or lambda functions to perform mapping, filtering, and reducing.

```python
>>> a = [1, 2, 3, 4, 5]
>>> print map(lambda x: x*2+1, a)
[3, 5, 7, 9, 11]

>>> g = lambda x: x%2 == 0
>>> print filter(g, a)
[2, 3]

>>> h = lambda x, y: x + y
>>> print reduce(h, a)
15
```
Using list comprehension:

```python
[expr  for  var  in  list  if  expr]
```

```python
>>> print [x*2 for x in a if x < 4]
[2, 4, 6]
```

```python
>>> def listOfSquares(a):
    return [x*x for x in a]
>>> listOfSquares([1, 2, 3])
[1, 4, 9]
```

```python
>>> d = {1: 'bob', 5: 'spam', 8: 'eggs', 22: 'spamspam'}
>>> [d[i] for i in d.keys() if i%2 == 0]
['eggs', 'spamspam']
```
Imperative Programming:

**Ex A.**
```python
intersect = []
for x in a:
    for y in b:
        intersect.append(y)
print intersect
```

**Ex B.**
```python
intersect = []
for x in a:
    if x in b:
        intersect.append(x)
print intersect
```

Functional Programming:

**Ex C.**
```python
intersect = filter(lambda x: x in b, a)
```

**Ex D.**
```python
intersect = [x for x in a if x in b]
```