Introduction to Programming in Python

Procedural Programming: Defining Functions

Outline

Function Definitions

Function Definitions

```
def <name>(<parameter1>, <parameter2>, ...):
    <statement>
    ...
```

```
return # to return from void functions

return <expression> # to return (with a value) from non-void functions
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y

>_

```
$ _
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y

>_

```
$ python3 square.py 13
```

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
1	"__main__"				

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
2	"__main__"				

>_

```
$ python3 square.py 13
```

```python
import stdio
import sys

def main():
    x = int(sys.argv[1])
    y = _square(x)
    stdio.writeln(y)

def _square(x):
    y = x * x
    return y

if __name__ == "__main__":
    main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
4-7	"__main__"				

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
9-11	"__main__"				

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
13	"__main__"				

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```python
import stdio
import sys

def main():
    x = int(sys.argv[1])
    y = _square(x)
    stdio.writeln(y)

def _square(x):
    y = x * x
    return y

if __name__ == "__main__":
    main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
14	"__main__"				

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```python
import stdio
import sys

def main():
    x = int(sys.argv[1])
    y = _square(x)
    stdio.writeln(y)

def _square(x):
    y = x * x
    return y

if __name__ == "__main__":
    main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
4	"__main__"				

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
5	"__main__"	13			

>_

```
$ python3 square.py 13
```

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
6	"__main__"	13			

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
9	"__main__"	13		13	

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
10	"__main__"	13		13	169

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```python
import stdio
import sys

def main():
    x = int(sys.argv[1])
    y = _square(x)
    stdio.writeln(y)

def _square(x):
    y = x * x
    return y

if __name__ == "__main__":
    main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
11	"__main__"	13		13	169

>_

```
$ python3 square.py 13
```

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
6	"__main__"	13	169		

>_

```
$ python3 square.py 13
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y
7	"__main__"	13	169		

>_

```
$ python3 square.py 13
169
```

Function Definitions · Control Flow

```
1  import stdio
2  import sys
3
4  def main():
5      x = int(sys.argv[1])
6      y = _square(x)
7      stdio.writeln(y)
8
9  def _square(x):
10     y = x * x
11     return y
12
13 if __name__ == "__main__":
14     main()
```

Variable Trace

line #	__name__	main::x	main::y	_square::x	_square::y

>_

```
$ python3 square.py 13
169
$ _
```

The scope of a function's local and parameter variables is limited to that function

The scope of a function's local and parameter variables is limited to that function

The scope of a variable defined in global code — known as a global variable — is limited to the `.py` file containing that variable

A function may designate an argument to be optional by specifying a default value for that argument

A function may designate an argument to be optional by specifying a default value for that argument

Example (computing $H_{n,r} = 1 + 1/2^r + 1/3^r + \cdots + 1/n^r$)

```
1 def harmonic(n, r = 1):
2     total = 0.0
3     for i in range(1, n + 1):
4         total += 1 / (i ** r)
5     return total
```

A function may designate an argument to be optional by specifying a default value for that argument

Example (computing $H_{n,r} = 1 + 1/2^r + 1/3^r + \cdots + 1/n^r$)

```
def harmonic(n, r = 1):
    total = 0.0
    for i in range(1, n + 1):
        total += 1 / (i ** r)
    return total
```

Calling `harmonic(5)` is the same as calling `harmonic(5, 1)`

If a function parameter refers to a mutable object, changing that object's value within the function also changes the object's value in the calling code

If a function parameter refers to a mutable object, changing that object's value within the function also changes the object's value in the calling code

Example

```
def exchange(a, i, j):
    temp = a[i]
    a[i] = a[j]
    a[j] = temp

a = [1, 2, 3, 4, 5]
exchange(a, 1, 3)
stdio.writeln(a)
```

writes

```
[1, 4, 3, 2, 5]
```

`harmonicredux.py`

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

```
>_ ~/workspace/ipp/programs

$ _
```

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

>_ ~/workspace/ipp/programs

```
$ python3 harmonicredux.py 10
```

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

>_ ~/workspace/ipp/programs

```
$ python3 harmonicredux.py 10
2.9289682539682538
$ _
```

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

>_ ~/workspace/ipp/programs

```
$ python3 harmonicredux.py 10
2.9289682539682538
$ python3 harmonicredux.py 1000
```

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

>_ ~/workspace/ipp/programs

```
$ python3 harmonicredux.py 10
2.9289682539682538
$ python3 harmonicredux.py 1000
7.485470860550343
$ _
```

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

>_ ~/workspace/ipp/programs

```
$ python3 harmonicredux.py 10
2.9289682539682538
$ python3 harmonicredux.py 1000
7.485470860550343
$ python3 harmonicredux.py 10000
```

harmonicredux.py

Command-line input	n (int)
Standard output	the nth harmonic number, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n) + 0.57721$

>_ ~/workspace/ipp/programs

```
$ python3 harmonicredux.py 10
2.9289682539682538
$ python3 harmonicredux.py 1000
7.485470860550343
$ python3 harmonicredux.py 10000
9.787606036044348
$ _
```

Examples · Harmonic Numbers

</> harmonicredux.py

```
import stdio
import sys

def main():
    n = int(sys.argv[1])
    stdio.writeln(_harmonic(n))

def _harmonic(n):
    total = 0.0
    for i in range(1, n + 1):
        total += 1 / i
    return total

if __name__ == "__main__":
    main()
```

</> harmonicredux.py

```
1  import stdio
2  import sys
3
4  def main():
5      n = int(sys.argv[1])
6      stdio.writeln(_harmonic(n))
7
8  def _harmonic(n):
9      total = 0.0
10     for i in range(1, n + 1):
11         total += 1 / i
12     return total
13
14 if __name__ == "__main__":
15     main()
```

When a program is imported as a library, the program's `__name__` attribute is not set to `"__main__"`

</> harmonicredux.py

```
1  import stdio
2  import sys
3
4  def main():
5      n = int(sys.argv[1])
6      stdio.writeln(_harmonic(n))
7
8  def _harmonic(n):
9      total = 0.0
10     for i in range(1, n + 1):
11         total += 1 / i
12     return total
13
14 if __name__ == "__main__":
15     main()
```

When a program is imported as a library, the program's `__name__` attribute is not set to `"__main__"`

>_ ~/workspace/ipp/programs

```
>>> _
```

</> harmonicredux.py

```
1  import stdio
2  import sys
3
4  def main():
5      n = int(sys.argv[1])
6      stdio.writeln(_harmonic(n))
7
8  def _harmonic(n):
9      total = 0.0
10     for i in range(1, n + 1):
11         total += 1 / i
12     return total
13
14 if __name__ == "__main__":
15     main()
```

When a program is imported as a library, the program's `__name__` attribute is not set to `"__main__"`

>_ ~/workspace/ipp/programs

```
>>> import harmonicredux
```

</> harmonicredux.py

```
1  import stdio
2  import sys
3
4  def main():
5      n = int(sys.argv[1])
6      stdio.writeln(_harmonic(n))
7
8  def _harmonic(n):
9      total = 0.0
10     for i in range(1, n + 1):
11         total += 1 / i
12     return total
13
14 if __name__ == "__main__":
15     main()
```

When a program is imported as a library, the program's `__name__` attribute is not set to `"__main__"`

>_ ~/workspace/ipp/programs

```
>>> import harmonicredux
>>> _
```

</> harmonicredux.py

```
1  import stdio
2  import sys
3
4  def main():
5      n = int(sys.argv[1])
6      stdio.writeln(_harmonic(n))
7
8  def _harmonic(n):
9      total = 0.0
10     for i in range(1, n + 1):
11         total += 1 / i
12     return total
13
14 if __name__ == "__main__":
15     main()
```

When a program is imported as a library, the program's `__name__` attribute is not set to `"__main__"`

>_ ~/workspace/ipp/programs

```
>>> import harmonicredux
>>> harmonicredux._harmonic(10)
```

</> harmonicredux.py

```
1  import stdio
2  import sys
3
4  def main():
5      n = int(sys.argv[1])
6      stdio.writeln(_harmonic(n))
7
8  def _harmonic(n):
9      total = 0.0
10     for i in range(1, n + 1):
11         total += 1 / i
12     return total
13
14 if __name__ == "__main__":
15     main()
```

When a program is imported as a library, the program's `__name__` attribute is not set to `"__main__"`

>_ ~/workspace/ipp/programs

```
>>> import harmonicredux
>>> harmonicredux._harmonic(10)
2.9289682539682538
>>> _
```

`couponcollectorredux.py`

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

>_ ~/workspace/ipp/programs

```
$ _
```

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

```
>_ ~/workspace/ipp/programs
$ python3 couponcollectorredux.py 1000
```

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

>_ ~/workspace/ipp/programs

```
$ python3 couponcollectorredux.py 1000
7462
$ _
```

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

>_ ~/workspace/ipp/programs

```
$ python3 couponcollectorredux.py 1000
7462
$ python3 couponcollectorredux.py 1000
```

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

~/workspace/ipp/programs

```
$ python3 couponcollectorredux.py 1000
7462
$ python3 couponcollectorredux.py 1000
9514
$ _
```

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

~/workspace/ipp/programs

```
$ python3 couponcollectorredux.py 1000
7462
$ python3 couponcollectorredux.py 1000
9514
$ python3 couponcollectorredux.py 1000000
```

couponcollectorredux.py

Command-line input	n (int)
Standard output	number of coupons one must collect before obtaining at least one of the n unique coupons

~/workspace/ipp/programs

```
$ python3 couponcollectorredux.py 1000
7462
$ python3 couponcollectorredux.py 1000
9514
$ python3 couponcollectorredux.py 1000000
13368303
$ _
```

</> couponcollectorredux.py

```
import stdarray
import stdio
import stdrandom
import sys

def main():
    n = int(sys.argv[1])
    stdio.writeln(_collect(n))

def _collect(n):
    count = 0
    collectedCount = 0
    isCollected = stdarray.create1D(n, False)
    while collectedCount < n:
        value = _getCoupon(n)
        count += 1
        if not isCollected[value]:
            collectedCount += 1
            isCollected[value] = True
    return count

def _getCoupon(n):
    return stdrandom.uniformInt(0, n)

if __name__ == "__main__":
    main()
```

playthattunedeluxe.py

Standard input	sound samples, each characterized by a pitch and a duration
Standard draw output	the sound

playthattunedeluxe.py

Standard input	sound samples, each characterized by a pitch and a duration
Standard draw output	the sound

>_ ~/workspace/ipp/programs

```
$ _
```

playthattunedeluxe.py

Standard input	sound samples, each characterized by a pitch and a duration
Standard draw output	the sound

~/workspace/ipp/programs

```
$ cat ../data/elise.txt
```

Examples · Play a Tune

playthattunedeluxe.py

Standard input	sound samples, each characterized by a pitch and a duration
Standard draw output	the sound

>_ ~/workspace/ipp/programs

```
$ cat ../data/elise.txt
7 .125
6 .125
7 .125
...
0 .25
$ _
```

Examples · Play a Tune

playthattunedeluxe.py

Standard input	sound samples, each characterized by a pitch and a duration
Standard draw output	the sound

~/workspace/ipp/programs

```
$ cat ../data/elise.txt
7 .125
6 .125
7 .125
...
0 .25
$ python3 playthattunedeluxe.py < ../data/elise.txt
```

Examples · Play a Tune

playthattunedeluxe.py

Standard input	sound samples, each characterized by a pitch and a duration
Standard draw output	the sound

>_ ~/workspace/ipp/programs

```
$ cat ../data/elise.txt
7 .125
6 .125
7 .125
...
0 .25
$ python3 playthattunedeluxe.py < ../data/elise.txt
$ _
```

Examples · Play a Tune

</> playthattunedeluxe.py 1/2

```
import math
import stdarray
import stdaudio
import stdio

def main():
    while not stdio.isEmpty():
        pitch = stdio.readInt()
        duration = stdio.readFloat()
        stdaudio.playSamples(_createRichNote(pitch, duration))
    stdaudio.wait()

def _createRichNote(pitch, duration):
    NOTES_ON_SCALE = 12
    CONCERT_A = 440.0
    hz = CONCERT_A * pow(2, pitch / NOTES_ON_SCALE)
    mid = _createNote(hz, duration)
    hi = _createNote(2 * hz, duration)
    lo = _createNote(hz / 2, duration)
    hiAndLo = _superpose(hi, lo, 0.5, 0.5)
    return _superpose(mid, hiAndLo, 0.5, 0.5)

def _createNote(hz, duration):
    SPS = 44100
    n = int(SPS * duration)
    note = stdarray.create1D(n + 1, 0.0)
    for i in range(n + 1):
        note[i] = math.sin(2 * math.pi * i * hz / SPS)
    return note

def _superpose(a, b, aWeight, bWeight):
    c = stdarray.create1D(len(a), 0.0)
    for i in range(len(a)):
        c[i] = a[i] * aWeight + b[i] * bWeight
    return c

```

playthattunedeluxe.py 2/2

```
36 if __name__ == "__main__":
37     main()
```

Filter, Lambda, and Map Functions

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

`filter(f, seq)` returns those items of `seq` for which `f(item)` is `True`

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

`filter(f, seq)` returns those items of `seq` for which `f(item)` is `True`

Example

```
>_ ~/workspace/ipp/programs
>>> _
```

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

`filter(f, seq)` returns those items of `seq` for which `f(item)` is `True`

Example

```
>_ ~/workspace/ipp/programs
>>> primes = filter(is_prime, range(11))
```

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

`filter(f, seq)` returns those items of `seq` for which `f(item)` is `True`

Example

```
>_ ~/workspace/ipp/programs

>>> primes = filter(is_prime, range(11))
>>> _
```

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

`filter(f, seq)` returns those items of `seq` for which `f(item)` is `True`

Example

```
>_ ~/workspace/ipp/programs

>>> primes = filter(is_prime, range(11))
>>> list(primes)
```

Filter, Lambda, and Map Functions

Functions in Python are first-class objects, meaning they can take functions as arguments and return functions as results

`filter(f, seq)` returns those items of `seq` for which `f(item)` is `True`

Example

```
>_ ~/workspace/ipp/programs

>>> primes = filter(is_prime, range(11))
>>> list(primes)
[2, 3, 5, 7]
>>> _
```

Filter, Lambda, and Map Functions

Filter, Lambda, and Map Functions

A lambda function is a “disposable” function that we can define just when we need it and then immediately throw it away after we are done using it

Filter, Lambda, and Map Functions

A lambda function is a “disposable” function that we can define just when we need it and then immediately throw it away after we are done using it

Example

```
>_ ~/workspace/ipp/programs
>>> _
```

Filter, Lambda, and Map Functions

A lambda function is a “disposable” function that we can define just when we need it and then immediately throw it away after we are done using it

Example

```
>_ ~/workspace/ipp/programs
>>> odds = filter(lambda x : x % 2 != 0, range(11))
```

Filter, Lambda, and Map Functions

A lambda function is a “disposable” function that we can define just when we need it and then immediately throw it away after we are done using it

Example

```
>_ ~/workspace/ipp/programs

>>> odds = filter(lambda x : x % 2 != 0, range(11))
>>> _
```

Filter, Lambda, and Map Functions

A lambda function is a “disposable” function that we can define just when we need it and then immediately throw it away after we are done using it

Example

```
>_ ~/workspace/ipp/programs
```

```
>>> odds = filter(lambda x : x % 2 != 0, range(11))
>>> list(odds)
```

Filter, Lambda, and Map Functions

A lambda function is a “disposable” function that we can define just when we need it and then immediately throw it away after we are done using it

Example

```
>_ ~/workspace/ipp/programs

>>> odds = filter(lambda x : x % 2 != 0, range(11))
>>> list(odds)
[1, 3, 5, 7, 9]
>>> _
```

Filter, Lambda, and Map Functions

Filter, Lambda, and Map Functions

`map(f, seq)` returns a list of the results of applying the function `f` to the items of `seq`

Filter, Lambda, and Map Functions

`map(f, seq)` returns a list of the results of applying the function `f` to the items of `seq`

Example

```
>_ ~/workspace/ipp/programs
>>> _
```

Filter, Lambda, and Map Functions

`map(f, seq)` returns a list of the results of applying the function `f` to the items of `seq`

Example

>_ ~/workspace/ipp/programs

```
>>> squares = map(lambda x : x ** 2, range(11))
```

Filter, Lambda, and Map Functions

`map(f, seq)` returns a list of the results of applying the function `f` to the items of `seq`

Example

>_ ~/workspace/ipp/programs

```
>>> squares = map(lambda x : x ** 2, range(11))
>>> _
```

Filter, Lambda, and Map Functions

`map(f, seq)` returns a list of the results of applying the function `f` to the items of `seq`

Example

>_ ~/workspace/ipp/programs

```
>>> squares = map(lambda x : x ** 2, range(11))
>>> list(squares)
```

Filter, Lambda, and Map Functions

`map(f, seq)` returns a list of the results of applying the function `f` to the items of `seq`

Example

```
>_ ~/workspace/ipp/programs
```

```
>>> squares = map(lambda x : x ** 2, range(11))
>>> list(squares)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> _
```