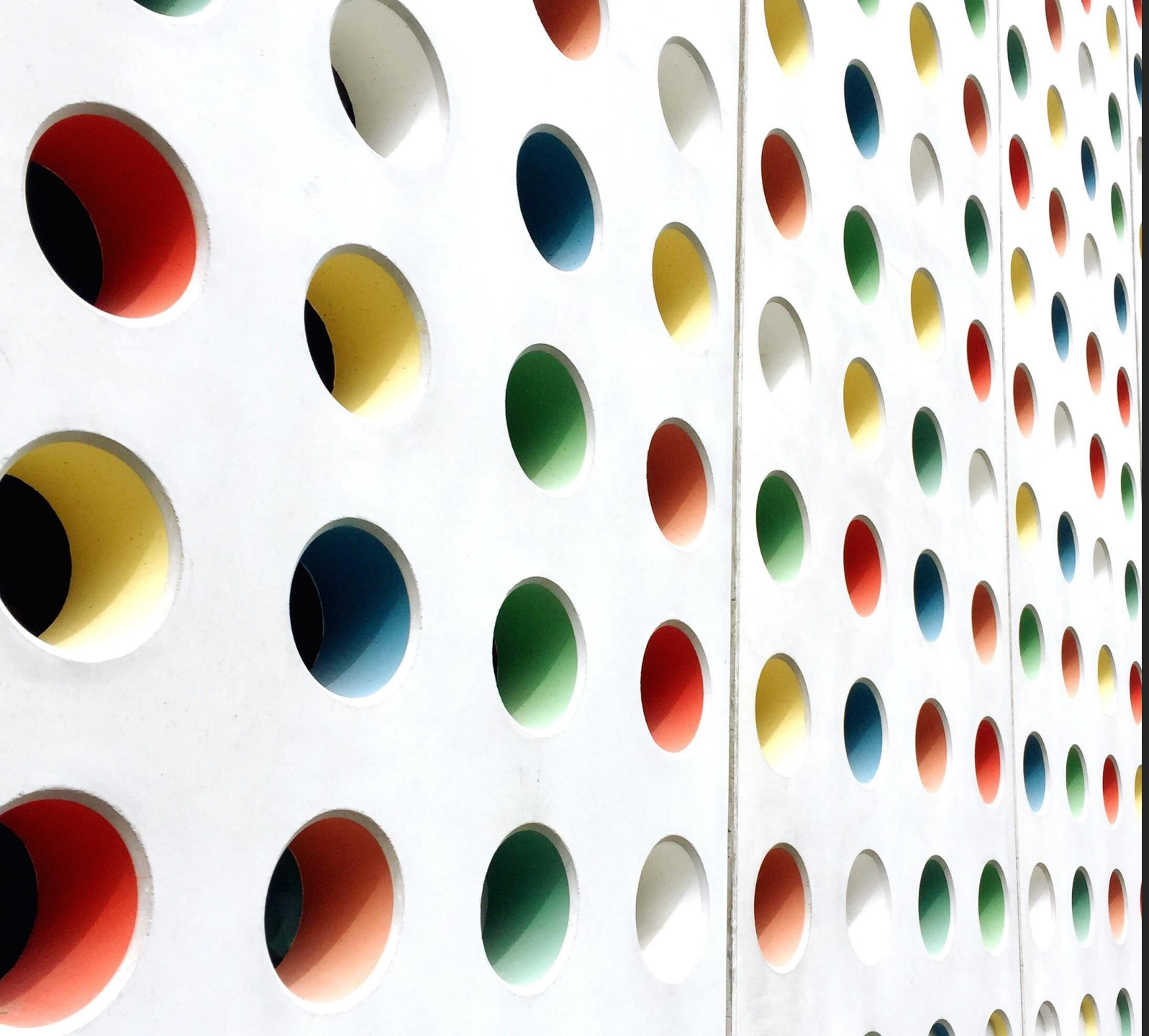


Thinking in Python

A QUICK INTRODUCTION AND SURVEY WITH EXAMPLES

Today's Agenda

- About the Presenter
- What This Course Isn't
- Python Basics
- Familiar Things
- Python-y Things
- Thinking in Python

About the Presenter

Drew DeNardo

drew@denardo.com / drew_denardo@cable.comcast.com

Twitter: @nt4cats

Reddit: nt4cats-reddit, /r/Crostini subreddit

- Certified Nerd
- Former Python Developer
- Currently a Suit on Comcast's Sales Technology Team

What This Course Isn’t

LET’S GET YOUR DISAPPOINTMENT OUT OF THE WAY

This course is not

SOMETHING THAT WILL MAKE MUCH SENSE IF YOU'RE NOT ALREADY A PROGRAMMER

A COMPREHENSIVE INTRODUCTION TO PYTHON SYNTAX AND STANDARD LIBRARIES

A DEEP-DIVE INTO ADVANCED TOPICS

This course is also not

EFFECTIVE AT PREVENTING BEE INFESTATIONS

EASY TO DANCE TO

MACHINE WASHABLE

Python Basics

NOT ENOUGH TO DO REAL WORK

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat Range.java
class Range {
        public static void main(String [] args) {
                for(int i=0;i<4;i++) {
                        System.out.println(i);
                }
                System.out.println("All done");
        }
}
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat Range.java
class Range {
        public static void main(String [] args) {
                for(int i=0;i<4;i++) {
                        System.out.println(i);
                }
                System.out.println("All done");
        }
}
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ java Range
0
1
2
3
All done
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Returns a sequence of the numbers between zero and the parameter

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

A colon like this signals the beginning of a code block.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Indentation signals what is in the block. Here this means the "print(i)" statement is the body of the for loop.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Blank lines are ignored.
Use them to make your
code easier to read.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

This line is indented less than the one above it. The outdent ends the above code block.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range.py
for i in range(4):
    print(i)

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 range.py
0
1
2
3
All done
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

These lines are all in one block which ends at the outdent below it.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

This line is an inner block nested inside the outer one.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Notice that this single outdented line ends all blocks above it that are indented more than it is.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 range2.py
0
---0,0
---0,1
---0,2
---0,3
1
---1,0
---1,1
---1,2
---1,3
2
---2,0
---2,1
---2,2
---2,3
3
---3,0
---3,1
---3,2
---3,3
All done
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 range2.py
0
---0,0
---0,1
---0,2
---0,3
1
---1,0
---1,1
---1,2
---1,3
2
---2,0
---2,1
---2,2
---2,3
3
---3,0
---3,1
---3,2
---3,3
All done
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Familiar Things

PYTHON SUPPORTS THINGS FROM YOUR FAVORITE LANGUAGES

Python is Object-Oriented

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Inline declaration of a String object

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat range2.py
for i in range(4):
    print(i)

    for j in range(4):
        print('---{},{}'.format(i,j))

print('All done')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

We're calling the format() method of that String object

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dir('This is a string')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
```

Built-in dir() function returns the methods and properties of an object

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dir('This is a string')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
```

Here we're implicitly declaring an instance of a String object.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dir('This is a string')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '_
_format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__ha
sh__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod_
_', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
 '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'ca
sefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_
map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'i
slower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartiti
on', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title'
, 'translate', 'upper', 'zfill']
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dir('This is a string')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
```

Methods that start and end with a double-underline (called "dunder methods") are, by convention, intended to be called only by the Python interpreter and standard library functions.

Python supports functional programming

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat func.py
class Dog:
    def eat(self, food):
        print('This dog eats the {}.'.format(food))

class Cat:
    def munch(self, food):
        print('This cat munches the {}.'.format(food))

    def get_munch(self):
        return self.munch

d = Dog()
c = Cat()

animal_feed = d.eat
animal_feed('popcorn')

animal_feed = c.get_munch()
animal_feed('steak')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Two classes defined. Each has a method that takes one argument.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat func.py
class Dog:
    def eat(self, food):
        print('This dog eats the {}.'.format(food))

class Cat:
    def munch(self, food):
        print('This cat munches the {}.'.format(food))

    def get_munch(self):
        return self.munch

d = Dog()
c = Cat()

animal_feed = d.eat
animal_feed('popcorn')

animal_feed = c.get_munch()
animal_feed('steak')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Methods can be assigned to variables

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat func.py
class Dog:
    def eat(self, food):
        print('This dog eats the {}.'.format(food))

class Cat:
    def munch(self, food):
        print('This cat munches the {}.'.format(food))

    def get_munch(self):
        return self.munch

d = Dog()
c = Cat()

animal_feed = d.eat
animal_feed('popcorn')

animal_feed = c.get_munch()
animal_feed('steak')

(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Methods can be returned from other methods

ddenardo@DREW-SB3: ~/py/thinking/source

```
ddenardo@DREW-SB3:~/py/thinking/source$ cat func.py
class Dog:
    def eat(self, food):
        print('This dog eats the {}.'.format(food))

class Cat:
    def munch(self, food):
        print('This cat munches the {}.'.format(food))

    def get_munch(self):
        return self.munch

d = Dog()
c = Cat()

animal_feed = d.eat
animal_feed('popcorn')

animal_feed = c.get_munch()
animal_feed('steak')

ddenardo@DREW-SB3:~/py/thinking/source$ python3 func.py
This dog eats the popcorn.
This cat munches the steak.
ddenardo@DREW-SB3:~/py/thinking/source$
```

Python supports closures

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat closure.py
def create_printer(msg):

    def printer():
        print(msg)

    return printer

anne = create_printer("I like games.")
mike = create_printer("I like books.")

anne()
mike()
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat closure.py
def create_printer(msg):

    def printer():
        print(msg)

    return printer

anne = create_printer("I like games.")
mike = create_printer("I like books.")

anne()
mike()
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

This is the last line of the create_printer() method

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat closure.py
def create_printer(msg):

    def printer():
        print(msg)

    return printer

anne = create_printer("I like games.")
mike = create_printer("I like books.")

anne()
mike()
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 closure.py
I like games.
I like books.
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Python-y Things

NOT UNIQUE, BUT EXAMPLE CORE ELEMENTS OF THE PYTHON VIBE

Python has a REPL

What's a REPL?

READ

EVAL

PRINT

LOOP

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>> s=7
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>> s=7
>>> s
7
>>> 
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>> s=7
>>> s
7
>>> s+2
9
>>> 
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dir('This is a string')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '_
_format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__ha
sh__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod_
_', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
 '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'ca
sefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_
map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'i
slower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartiti
on', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title'
, 'translate', 'upper', 'zfill']
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> s={}
>>> for o in dir('foo'):
...     t = type(getattr('foo',o))
...     if t in s:
...         x = s[t]
...         x = x + 1
...         s[t] = x
...     else:
...         s[t] = 1
...
>>> s
{<class 'method-wrapper'>: 22, <class 'type'>: 1, <class 'builtin_function_or_method'>: 54
, <class 'str'>: 1}
>>>
```

Thinking in Python

EXAMPLES OF DOING THINGS THE PYTHON WAY

Python aims to make a programmer's job easier

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

The range() method supports a starting point (inclusive) and an ending point (exclusive).

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 evens.py
[2, 4, 6, 8, 10]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 evens.py
[2, 4, 6, 8, 10]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [i*2 for i in range(1,6)]
[2, 4, 6, 8, 10]
>>> 
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 evens.py
[2, 4, 6, 8, 10]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [i*2 for i in range(1,6)]
[2, 4, 6, 8, 10]
>>> 
```

This is a sequence that will be iterated upon

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 evens.py
[2, 4, 6, 8, 10]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [i*2 for i in range(1,6)]
[2, 4, 6, 8, 10]
>>>
```

This is the variable that will be assigned each value in the sequence one at a time.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 evens.py
[2, 4, 6, 8, 10]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [i*2 for i in range(1,6)]
[2, 4, 6, 8, 10]
>>>
```

This is an expression that will be run to produce the values in the output list.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat evens.py
evens=[]

for i in range(1,6):
    evens.append(i*2)

print(evens)
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 evens.py
[2, 4, 6, 8, 10]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [i*2 for i in range(1,6)]
[2, 4, 6, 8, 10]
>>> 
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [k for k in range(1,12)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [k for k in range(1,12)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> [k for k in range(1,12) if k%2==0]
[2, 4, 6, 8, 10]
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [k for k in range(1,12)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> [k for k in range(1,12) if k%2==0]
[2, 4, 6, 8, 10]
>>>
```

This filter limits the values that are assigned to the variable *k*

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [k for k in range(1,12)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> [k for k in range(1,12) if k%2==0]
[2, 4, 6, 8, 10]
>>> [(a,b) for a in range(0,5) for b in range(10,15)]
[(0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (1, 10), (1, 11), (1, 12), (1, 13), (1, 14),
 (2, 10), (2, 11), (2, 12), (2, 13), (2, 14), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14),
 (4, 10), (4, 11), (4, 12), (4, 13), (4, 14)]
>>>
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [k for k in range(1,12)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> [k for k in range(1,12) if k%2==0]
[2, 4, 6, 8, 10]
>>> [(a,b) for a in range(0,5) for b in range(10,15)]
[(0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (1, 10), (1, 11), (1, 12), (1, 13), (1, 14),
 (2, 10), (2, 11), (2, 12), (2, 13), (2, 14), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14),
 (4, 10), (4, 11), (4, 12), (4, 13), (4, 14)]
>>>
```

Adjacent *for {x} in {y}* clauses result in a cartesian product.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> [k for k in range(1,12)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> [k for k in range(1,12) if k%2==0]
[2, 4, 6, 8, 10]
>>> [(a,b) for a in range(0,5) for b in range(10,15)]
[(0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (1, 10), (1, 11), (1, 12), (1, 13), (1, 14),
 (2, 10), (2, 11), (2, 12), (2, 13), (2, 14), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14),
 (4, 10), (4, 11), (4, 12), (4, 13), (4, 14)]
>>>
```

```
    System.out.print("Please Enter The Name Of A File Or "
        + "Directory, or Type Quit To Exit: ");
    String nameOfFile = keys.nextLine().trim(); // get the User input.
    if (nameOfFile.equalsIgnoreCase("quit")) {  // check for exit condition.
      break;
    }
    File f = new File(nameOfFile);              // Construct a File.
    if (f.exists()) {                           // Does it exist?
      if (f.isFile() && f.canRead()) {          // is it a File and can I read it?
        Scanner input = null;
        try {
          input = new Scanner(f);               // The Scanner!
          while (input.hasNextLine()) {
            String contents = input.nextLine();
            System.out.println(contents);       // Print the lines.
          }
        } catch (FileNotFoundException e) {
          e.printStackTrace();
        } finally {
          if (input != null) {
            input.close();                      // Close the file scanner.
          }
        }
      } else if (f.isDirectory()) {             // No, it's a directory!
        try {
          System.out.println("File "
              + f.getCanonicalPath()
              + " is a directory");
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }
  }
}
```

```java
    System.out.print("Please Enter The Name Of A File Or "
        + "Directory, or Type Quit To Exit: ");
    String nameOfFile = keys.nextLine().trim(); // get the User input.
    if (nameOfFile.equalsIgnoreCase("quit")) {  // check for exit condition.
      break;
    }
    File f = new File(nameOfFile);              // Construct a File.
    if (f.exists()) {                           // Does it exist?
      if (f.isFile() && f.canRead()) {          // is it a File and can I read it?
        Scanner input = null;
        try {
          input = new Scanner(f);               // The Scanner!
          while (input.hasNextLine()) {
            String contents = input.nextLine();
            System.out.println(contents);       // Print the lines.
          }
        } catch (FileNotFoundException e) {
          e.printStackTrace();
        } finally {
          if (input != null) {
            input.close();                      // Close the file scanner.
          }
        }
      } else if (f.isDirectory()) {             // No, it's a directory!
        try {
          System.out.println("File "
              + f.getCanonicalPath()
              + " is a directory");
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }
  }
}
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat reading.py
words = open('/etc/dictionaries-common/words','r')

lines = 0

try:
    for line in words:
        lines = lines + 1
finally:
    try:
        words.close()
    except:
        # not much we can do if this fails
        pass

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat reading.py
words = open('/etc/dictionaries-common/words','r')

lines = 0

try:
    for line in words:
        lines = lines + 1
finally:
    try:
        words.close()
    except:
        # not much we can do if this fails
        pass

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Opens a file at the given path, second argument indicates “read” mode, and return a File object.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat reading.py
words = open('/etc/dictionaries-common/words','r')

lines = 0

try:
    for line in words:
        lines = lines + 1
finally:
    try:
        words.close()
    except:
        # not much we can do if this fails
        pass

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

File objects are iterable, they return one line of text from the file per iteration.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat reading.py
words = open('/etc/dictionaries-common/words','r')

lines = 0

try:
    for line in words:
        lines = lines + 1
finally:
    try:
        words.close()
    except:
        # not much we can do if this fails
        pass

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

"pass" is a no-op in python. It does nothing.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat reading.py
words = open('/etc/dictionaries-common/words','r')

lines = 0

try:
    for line in words:
        lines = lines + 1
finally:
    try:
        words.close()
    except:
        # not much we can do if this fails
        pass

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 reading.py
102401 lines.
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat with.py
lines=0

with open('/etc/dictionaries-common/words','r') as f:
    for line in f:
        lines = lines + 1

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat with.py
lines=0

with open('/etc/dictionaries-common/words','r') as f:
    for line in f:
        lines = lines + 1

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

The **with** keyword signals the use of a critical resource.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat with.py
lines=0

with open('/etc/dictionaries-common/words','r') as f:
    for line in f:
        lines = lines + 1

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

The expression that follows **with** produces the critical resource.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat with.py
lines=0

with open('/etc/dictionaries-common/words','r') as f:
    for line in f:
        lines = lines + 1

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

The variable that follows the **as** keyword is assigned the critical resource.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat with.py
lines=0

with open('/etc/dictionaries-common/words','r') as f:
    for line in f:
        lines = lines + 1

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

When the block above this ends the Python environment will make sure that the critical resource is cleaned up.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat with.py
lines=0

with open('/etc/dictionaries-common/words','r') as f:
    for line in f:
        lines = lines + 1

print('{} lines.'.format(lines))
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python3 with.py
102401 lines.
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat infinite.py
# This is an example of a program that won't work!

def infinite_sequence():
    infinite = []
    i = 0
    while True:
        infinite.append(i)
        i = i + 1

    return infinite

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat infinite.py
# This is an example of a program that won't work!

def infinite_sequence():
    infinite = []
    i = 0
    while True:
        infinite.append(i)
        i = i + 1

    return infinite

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Start with an empty list

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat infinite.py
# This is an example of a program that won't work!

def infinite_sequence():
    infinite = []
    i = 0
    while True:
        infinite.append(i)
        i = i + 1

    return infinite

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Here is a trusty infinite loop …

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat infinite.py
# This is an example of a program that won't work!

def infinite_sequence():
    infinite = []
    i = 0
    while True:
        infinite.append(i)
        i = i + 1

    return infinite

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

This line will never execute

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat generator.py
def infinite_sequence():
    i = 0
    while True:
        yield i
        i = i + 1

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat generator.py
def infinite_sequence():
    i = 0
    while True:
        yield i
        i = i + 1

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

We still have the same while clause that gives us an infinite loop.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat generator.py
def infinite_sequence():
    i = 0
    while True:
        yield i
        i = i + 1

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

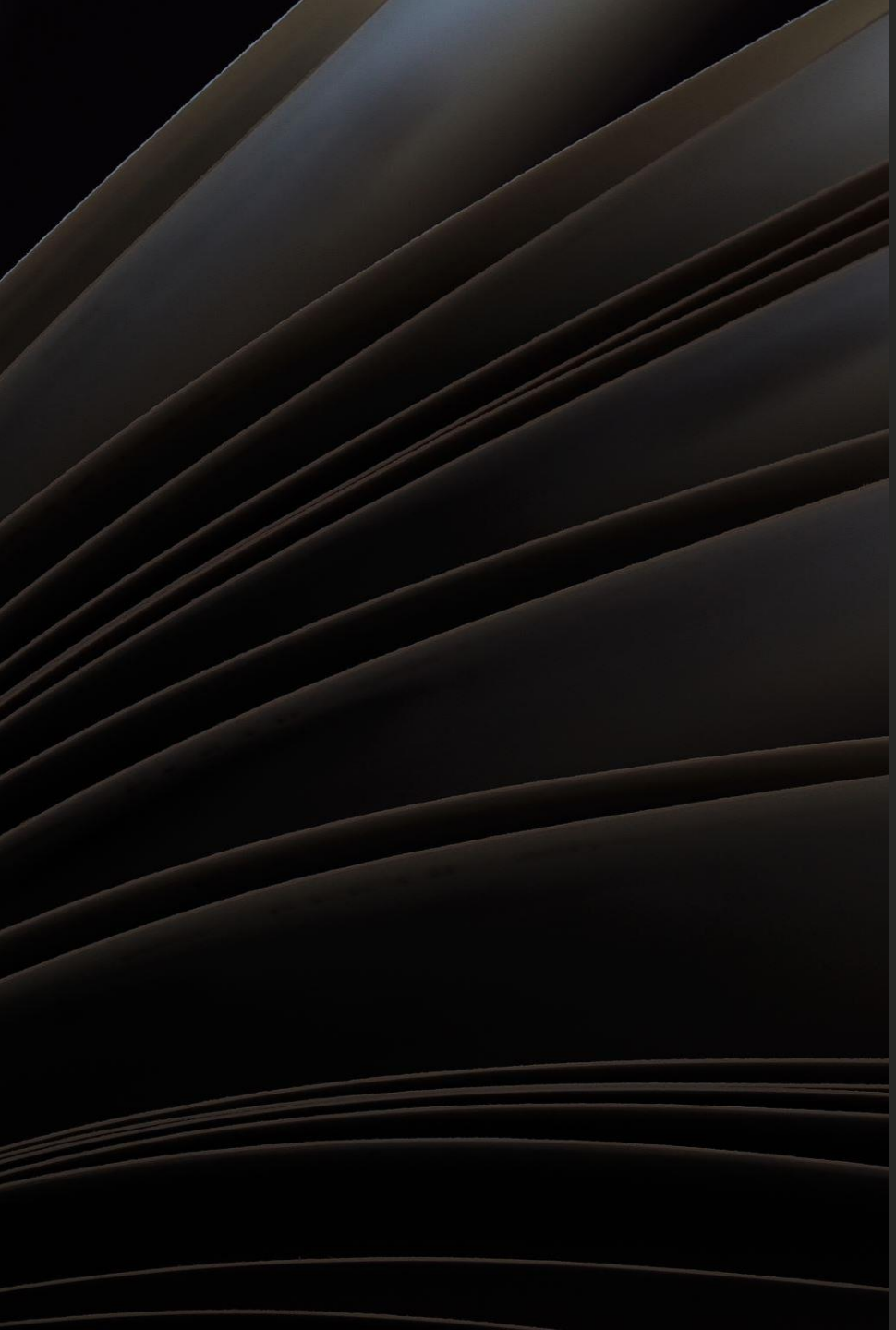
The yield keyword pauses the execution of the current method, saves its state, and returns the argument.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat generator.py
def infinite_sequence():
    i = 0
    while True:
        yield i
        i = i + 1

for x in infinite_sequence():
    if x < 5:
        print(x)
    else:
        break
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ python generator.py
0
1
2
3
4
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Use the Python Data Model in Your Classes



Attribution

The code for this example is taken with the Author's permission from *Fluent Python* by Luciano Ramalho (O'Reilly). Copyright 2015 Luciano Ramalho, 978-1-491-94600-8. This code is publicly available at the GitHub repository for the book: https://github.com/fluentpython/example-code.

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat frenchdeck.py
import collections

Card = collections.namedtuple('Card', ['rank', 'suit'])

class FrenchDeck:
    ranks = [str(n) for n in range(2, 11)] + list('JQKA')
    suits = 'spades diamonds clubs hearts'.split()

    def __init__(self):
        self._cards = [Card(rank, suit) for suit in self.suits
                                        for rank in self.ranks]

    def __len__(self):
        return len(self._cards)

    def __getitem__(self, position):
        return self._cards[position]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat frenchdeck.py
import collections

Card = collections.namedtuple('Card', ['rank', 'suit'])

class FrenchDeck:
    ranks = [str(n) for n in range(2, 11)] + list('JQKA')
    suits = 'spades diamonds clubs hearts'.split()

    def __init__(self):
        self._cards = [Card(rank, suit) for suit in self.suits
                                        for rank in self.ranks]


    def __len__(self):
        return len(self._cards)

    def __getitem__(self, position):
        return self._cards[position]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

Two lists, appended

ddenardo@DREW-SB3: ~/py/thinking/source

```
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$ cat frenchdeck.py
import collections

Card = collections.namedtuple('Card', ['rank', 'suit'])

class FrenchDeck:
    ranks = [str(n) for n in range(2, 11)] + list('JQKA')
    suits = 'spades diamonds clubs hearts'.split()

    def __init__(self):
        self._cards = [Card(rank, suit) for suit in self.suits
                                        for rank in self.ranks]

    def __len__(self):
        return len(self._cards)

    def __getitem__(self, position):
        return self._cards[position]
(thinking) ddenardo@DREW-SB3:~/py/thinking/source$
```

The split() method takes a delimited list of strings and returns a list of strings. Space is the default delimiter.

Live Examples

We'll drop into the REPL now and show you examples of manipulating this deck of cards using standard built-ins.

Q&A

WHAT MORE WOULD YOU LIKE TO KNOW?

Resources

WHERE TO GO TO LEARN MORE

Source & Presentation

Available at my public GitHub site:

https://github.com/nt4cats/thinking-in-python

Websites

The official source: https://docs.python.org/3/

Great resource for learning Python: https://www.programiz.com/python-programming

Online version of one of the original Python 3 books: https://diveintopython3.net/index.html

Books

Making the most of Python3: *Fluent Python* by Luciano Ramalho (O'Reilly). Copyright 2015 Luciano Ramalho, 978-1-491-94600-8.

Recipe book of Python idioms: *Python Cookbook: Recipes for Mastering Python 3* by David Beazley & Brian K. Jones (O'Reilly). Copyright 2013 David Beazley & Brian K. Jones. 978-1-449-34037-7.