Introduction to Python

Functional Programming in Python

October 27, 2016

Lambda Calculus

- **Idea:** Create a syntax to describe computation and study it formally as an abstract process.

Lambda Calculus

- **Idea:** Create a syntax to describe computation and study it formally as an abstract process.
- First formulated by Alonzo Church just as Turing was inventing Turing machines.

Lambda Calculus

- **Idea:** Create a syntax to describe computation and study it formally as an abstract process.
- First formulated by Alonzo Church just as Turing was inventing Turing machines.
- Formally equivalent to Turing machines (proved by Turing). Led to the Church-Turing Thesis.

Lambda Calculus in the Real World

- **1958:** Lisp invented - a realisation of the Lambda Calculus. Nowadays: Common Lisp, Scheme, Clojure

Lambda Calculus in the Real World

- **1958:** Lisp invented - a realisation of the Lambda Calculus. Nowadays: Common Lisp, Scheme, Clojure
- **1970s:** ML invented at University of Edinburgh as a language to prove theorems with. Nowadays: OCaml, Standard ML.

Lambda Calculus in the Real World

- **1958:** Lisp invented - a realisation of the Lambda Calculus. Nowadays: Common Lisp, Scheme, Clojure
- **1970s:** ML invented at University of Edinburgh as a language to prove theorems with. Nowadays: OCaml, Standard ML.
- **1987:** Haskell language invented, taught at Edinburgh to first years

Lambda Calculus in the Real World

- **1958:** Lisp invented - a realisation of the Lambda Calculus. Nowadays: Common Lisp, Scheme, Clojure
- **1970s:** ML invented at University of Edinburgh as a language to prove theorems with. Nowadays: OCaml, Standard ML.
- **1987:** Haskell language invented, taught at Edinburgh to first years
- **Present day:** Erlang, Python, C++ (ish), Swift and many more!

Concepts

- **Functions are First-Class Citizens:** Functions are just examples of data - they can be parameters in functions and return values to functions

Concepts

- **Functions are First-Class Citizens:** Functions are just examples of data - they can be parameters in functions and return values to functions
- **Pure functions:** Functions have no side effects. In particular every function is idempotent.

Concepts

- **Functions are First-Class Citizens:** Functions are just examples of data - they can be parameters in functions and return values to functions
- **Pure functions:** Functions have no side effects. In particular every function is idempotent.
- **Recursion:** Because of the “no mutation” philosophy, recursion is always preferred over iteration.

Concepts

- **Functions are First-Class Citizens:** Functions are just examples of data - they can be parameters in functions and return values to functions
- **Pure functions:** Functions have no side effects. In particular every function is idempotent.
- **Recursion:** Because of the “no mutation” philosophy, recursion is always preferred over iteration.
- **Python is not a pure-functional language. It’s up to you what you use and ignore from the above**

Demo

Demo Time!

Helper Functions

- You can define functions inside functions. These will be inaccessible to the outside world.

```
  def f(n):
   def g(m):
return m*m #latex won't let me indent  :(
   return g(n)
```

- This is useful for de-cluttering your code and hiding functionality you don't want to be public.

Closures

- Helper functions are more powerful than at first they seem. Remember functions can be return values as well!

Closures

- Helper functions are more powerful than at first they seem. Remember functions can be return values as well!
- When you return a function Python will *capture* local variables needed to use later.

```
  def powerFactory(n):
   def g(x):
return x ** n
   return g
```

Closures

- Helper functions are more powerful than at first they seem. Remember functions can be return values as well!
- When you return a function Python will *capture* local variables needed to use later.

```
  def powerFactory(n):
   def g(x):
return x ** n
   return g
```

- **Demo Time!**

Lambdas

- Sometimes it can be annoying to come up with names for functions you will only use once.

Lambdas

- Sometimes it can be annoying to come up with names for functions you will only use once.
- Use Lambdas to get around this

```
def powerFactory(n):
 return (lambda x: x ** n)
```

Demo

Demo Time!

Introduction

- Map, Reduce and Filter are three common functions from functional programming.
- Each function acts on a list. Since these are functional they do not mutate the list!

Introduction

- Map, Reduce and Filter are three common functions from functional programming.
- Each function acts on a list. Since these are functional they do not mutate the list!
- The names have since become famous because of the 'MapReduce' framework invented by Google for Big Data calculations

Map

- Map takes a function of one variable and a list and acts componentwise on the list.

Map

- Map takes a function of one variable and a list and acts componentwise on the list.
- For example:

```
x = [1,2,3]
print map(lambda z: z ** 2, x)
# prints [1, 4, 9]
```

Reduce

- Reduce takes a function of two variables, a list and an initial value and 'folds' down the list.

Reduce

- Reduce takes a function of two variables, a list and an initial value and 'folds' down the list.
- For example:

```
x = [1,2,3]
print reduce(lambda y,z: y + z, x, 0)
# prints 6
```

Filter

- Filter takes a function of one variable that returns True or False, a list and returns the list with only those elements that the function returns True on.

Filter

- Filter takes a function of one variable that returns True or False, a list and returns the list with only those elements that the function returns True on.
- For example:

```
x = [1,2,3]
print reduce(lambda z: z \% 2, x)
# prints [1,3]
```