# **Programming Languages Final Project**

Tracy Cheng

Andrew Lau

#### **Python Closure**

- Computer Class
  - -getSpecs = dictionary of data
    - •Name, CPU, # of CPU, Motherboard, RAM, Harddrive, PSU
- SuperComp and Gaming Class
  - -Inherits from Computer Class
  - -Have additional specs
- Program Class

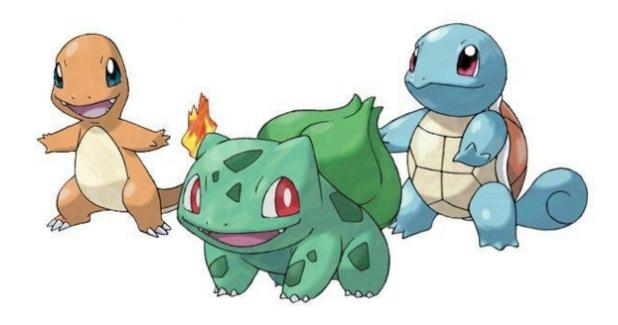
### **Python Closure**

- Data is kept in dictionary in getSpecs
- •Cf is a function within getSpecs that is essentially a setter and a getter
- getSpecs is set to run
- •Run is called to get and set data

#### **Testing Python Closure**

```
(exec 'import test;
c1=test.Computer(); //create object
c1.run("$name")("I am a computer"); //set
toReturn = c1.run("name")') //get
(exec 'import test; c1=test.Computer(); c1.run("name"); c1.run("$name")("I am a computer"); toReturn = c1.run("name")')
```

## Anyone remember these guys?



#### **Java Stream Operations**

- Created Pokemon and Strengths classes
  - Pokemon class has Id, Name, type1, type2
    - type1 and type2 are the pokemon type of the pokemon (Fire, water, grass, etc.)
  - Strengths class has Type, Str1, Str2, Str3, Str4
    - Type is strong against Str1, Str2, Str3, and Str4
- Created pokemonparse.java
  - Functions: loadData(), getIDs(), getAllofType(), getStrong(), and getStrongPokemons()
  - loadData() Loads data by reading files pokemon and strength and storing them in pokedex and strengths
  - The other functions use Java Stream Operations to do List Comprehension

#### **Testing Java Stream Operations**

- In mini\_ply:
  - (exec 'import pokemonparse;
  - o pokemonparse.loadData(); //loads the data from the files, creating objects as it goes
  - pokemonparse.getIDs(["Pikachu","Beedrill"]) //goes through pokedex and returns the IDs of a list of pokemon names
  - 0
- (exec 'import pokemonparse; pokemonparse.loadData(); pokemonparse.getIDs(["Pikachu"," Beedrill"])')

#### Python Lambda and List Comprehension

- Used Python Lambda and List Comprehension to implement FindStrongPokemon function in mini-lisp
- The function takes in 3 parameters: List(pokemon), List(strengths), List(strongtype)
- Created a ListLoader function to import ListMaker.py and return the resulting list
- Created ListMaker.py to make a list when given a file name
- Added code to the env() to implement FindStrongPokemon (see next slide)

Input format: (FindStrongPokemon (ListMaker 'databaseofpokemon') (ListMaker 'database of strengths') '('pokemontype'))

#### **Exploring the Lambda Function**

Inside lis.py

'FindStrongPokemon':

lambda pokedex, strengths, strongType:

[[pokemontype, sorted([p[1] for p in pokedex if p[2] == pokemontype or p[3] == pokemontype])]

for pokemontype in sorted( $\{s[0] \text{ for s in strengths if } s[1] == strongType[0] \text{ or } s[2] == strongType[0] \text{ or } s[3] == strongType[0] \text{ or } s[4] == strongType[0]\})]$ 

#### **Testing Python Lambda and List Comprehension**

- (FindStrongPokemon)
- (ListLoader 'pokemon') //loads pokemon into a list, list acts as pokedex parameter
- (ListLoader 'strength') //loads strength into a list, list acts as strengths parameter
- '(Grass) //list acts as strongtype parameter

(FindStrongPokemon (ListLoader 'pokemon')(ListLoader 'strength')'(Grass))

#### **Swift**

```
let apples = 3
```

let oranges = 5

let appleSummary = "I have \(apples) apples."

let fruitSummary = "I have \(apples + oranges) pieces of fruit."

print(fruitSummary)

