DATA ANALYTICS REFERENCE DOCUMENT	
Document Title:	52960 - Multi-Paradigm Programming
Document No.:	1569071117
Author(s):	Gerhard van der Linde, Rita Raher
Contributor(s):	

REVISION HISTORY

Revision	Details of Modification(s)	Reason for modification	Date	By
0	Draft release	Document description here	2019/09/21 13:05	Gerhard van der Linde

52960 - Multi-Paradigm Programming

Goals

- What is a programming paradigm?
- Why are there different paradigms?
- How does it relate the notions of State & Abstraction?

What is a Programming Paradigm?

- The term programming paradigm is the style or way of thinking about and approaching problems.
- The chosen paradigm affects how the code is written and structured.
- It can heavily influence how one thinks about the problem being solved
- Some problems map more easily to a particular paradigm
- Each paradigm has it's advocates and detractors, advantages and disadvantages etc.
- Different Lanquage is not the same thing as a different Paradigm

MIPS Assembly

```
LUI R1, #1
LUI R2, #2
DADD R3, R1, R2
```

C or Java

(In fact this is syntactically valid in a lot of languages)

```
x = 1 + 2;
```

- Both examples show the addition of two numbers
- MIPS is very low level, one step above binary
- C is higher level, though most would consider C to be a low level language
- Both examples follow the same paradigm, such as it can be in a “one liner”

Abstraction

- In software engineering and computer science, abstraction is:
    - the process of removing **physical**, **spatial**, or **temporal** details or attributes in the study of objects or systems in order to focus attention on details of higher importance, it is also very similar in nature to the process of generalization;
- the creation of abstract concept-objects which are created by mirroring common features or attributes from various non-abstract objects or systems of study - the result of the process of abstraction.
- It is one of the most important concepts in Software Development
- So much of Computer Science is about Abstraction

Examples of Abstraction

- Concurrency Control
- Memory Management
- Virtual Machine
- Operating Systems
- Drivers
- APIs

- All Programming languages and Paradigms are attempts to abstract low-level details to allow the programmer to think about and solve problems at a higher level (or perhaps a different level!)

- Computers understand operations at a very low level
- i.e. Moving bits from one place to another
- Of course we wanted to do things at a higher level than bitwise operations
- Have a look at the following operation:
  a := (1 + 2) * 5
- so it's “one plus two multiplied by five”
- The low level steps needed to
    - carry out this evaluation
    - return the result (15)
    - and perform assignment (to a)
- are actually quite complex
    - recall it's a rock we tricked into thinking

What is state?

- A program can store data in variables, which map to storage locations in memory. The contents of these memory locations, at any

given point in the program's execution, is called the program's state.

- State effects the behaviour of the program.
- The more state the more unpredictable the program
- “In programming mutable state is evil”
- Some paradigms would seek to do away with it completely.
- In others it is intrinsic, OOP without mutable state is not possible.

```
var total = 0;
var a = 1;
var b = 5;
total = a + b
print total;
```

- In the beginning total is 0
- it's state is modified
- then printed
- No problems here but…
    - this can be complicated by control flow structures dependant on the value of variables, unpredictable values entered by users or coming from stored data

Various Programming Paradigms

- Imperative / Procedural
- Functional
- Object-oriented
- Declarative
- Data Flow

We shall have a brief look at each of these….

Sources

- https://www.computerhope.com/jargon/a/al.htm
- https://en.wikipedia.org/wiki/Abstraction_(computer_science)
- https://en.wikipedia.org/wiki/C_(programming_language)

Week 3 - Imperative: Procedural

Goals of this session

- Imperative programming
    - Procedural programming
        - Python Example
        - Procedural Programming Example
        - Imperative Programming Example
- Sources

Goals

- To Understand…
    - What is Imperative programming?
    - What is Procedural program?
    - How a procedural program is structured

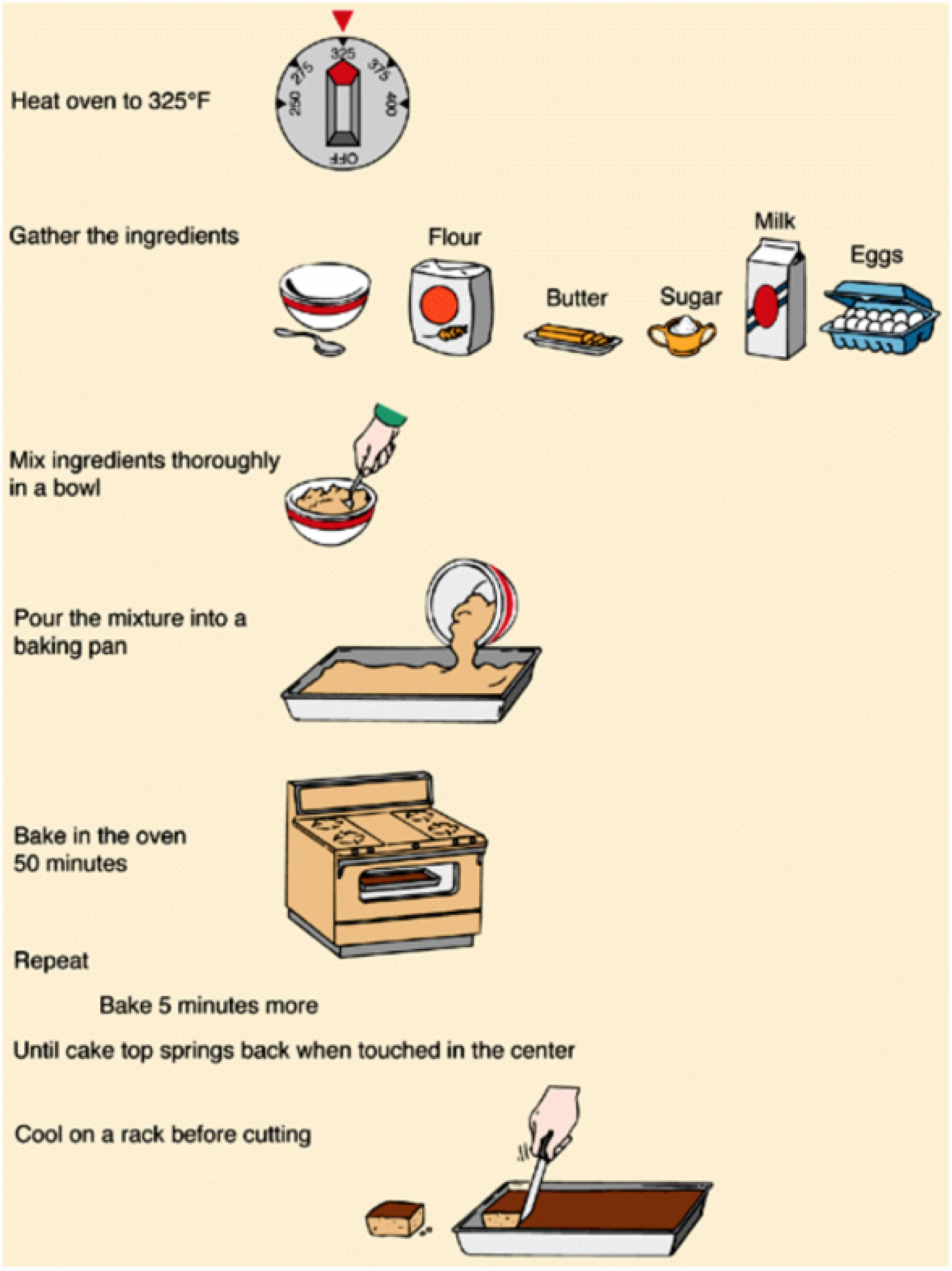
Imperative Programming

Imperative I

- Programming with an explicit sequence of commands that update state
- Says **how** to do something

- consider baking a cake (just good life advice really)
    - An imperative program says how to do something in the correct sequence it should be done
        - therefore order of execution(the order in which each statement is executed) is important.
        - Obviously you cannot add candles if you didn't bake the cake right?
        - You cannot eat no cake!

Imperative II


Heat oven to 325°F
325
275
375
250
400
OFF
Gather the ingredients
Flour
Butter
Sugar
Milk
Eggs
Mix ingredients thoroughly in a bowl
Pour the mixture into a baking pan
Bake in the oven 50 minutes
Repeat
Bake 5 minutes more
Until cake top springs back when touched in the center
Cool on a rack before cutting

Imperative III

Very General Structure First **do this** and next **do that**, then **repeat this 10 times**, then **finish**

Imperative IV

- Is simply Imperative programming with procedure calls
  - AKA methods or functions or sub-routines
- To avoid repetition and to provide structure early programmers realized they should group instructions together into re-usable elements called procedures which can then be called when needed during program execution
- Imperative did provide a means of non-linear execution a “go-to” but it was messy.
  - This was one of the first advances in programming maintainability.
  - The idea was first implemented in ALGOL in the 1950’s
- Mostly when someone say they are doing Imperative Programming they are really doing Procedural Programming

Imperative V

Listing 1: Very Simple Function in Python

```
def my_function():
   print("Hello from a function")

my_function()
```

Imperative VI

Listing 2: Slightly less simple Function in Python

```
def print_list(food):
   for x in food:
     print(x)

fruits = ["apple", "banana", "cherry"]

print_list(fruits)
```

Imperative VII

Listing 3: C Language Example (Procedural)

```
#print numbers from 1 to 10
# include <stdio.h>

int main(){
   int i;
   for (i=1; i<11; ++i)
   {
   printf("%d ", i);
   }
   return 0;
}
```

Listing 3: Python Language Example (Procedural)

```
def myFunction():
   a = 0
   for a in range(0, 10):
      a += 1 # Same as a = a + 1
      print (a, sep=' ', end=' ', flush=True)


myFunction()
```

- Execution begins at **main()** the programs entry point
- Proceeds linearly (kinda)
- Loop repeats until condition is false
- **printf()** is a function taking args, defined as part of standard C
- Program terminates returing exit code 0

Imperative VIII

Listing 4: Imperative with go-to

```
result = []
i = 0
start:
  numPeople = length(people)
  if i >= numPeople goto finished
  p = people[i]
  nameLength = length(p.name)
  if nameLength <= 5 goto nextOne
  upperName = toUpper(p.name)
  addToList(result, upperName)
nextOne:
  i = i + 1
  goto start
finished:
  return sort(result)
```

Imperative IX

- Imperative Programs often compile to binary executables that run more efficiently since all CPU instructions are themselves imperative statements
- Imperative approaches and “lower level” languages are often used where efficiency and code-footprint are important such as in embedded systems.
  - “Low Level” refers to the closeness to the machine with regard to the level of abstraction
  - An approach like declarative programming would be more “high level”

Imperative X

- Procedural Languages
  - C
  - Python
  - ALGOL
  - JavaScript
  - PHP

Sources

- https://www.computerhope.com/jargon/i/imp-programming.htm

- https://en.wikipedia.org/wiki/Abstraction_(computer_science)
- https://www.w3resource.com/c-programming-exercises/basic-algo/index.php

Week 3 - The C Programming language

- **Goals of this session**
- **The C Programming Language**
    - Structuring Data in C
    - Comparison with Python
    - Installing C on Windows
    - C Practice Questions
- **Sources**
- To Understand...
    - The basics of the C programming language
    - How to write a procedural program in C

The C programming Language

The C programming Language I

- general-purpose & procedural computer programming language
- Supports structured programming, lexical variable scope, and recursion
- Static type system prevents unintended operations.
- By design, C provides constructs that **map efficiently** to typical **machine instructions**
    - Has found lasting use in applications previously coded in **assembly language**.
    - Including operating systems and application software for diverse
    - platforms from supercomputers to embedded systems.
    - It has been around since 1972
    - Was developed at Bell Labs

The C programming Language II

Listing 1: Assembly Code for Hello World

```
global _main
extern _printf

section .text
_main:
  push message
  call _printf
  add esp, 4
  ret
message:
  db 'Hello, World!', 10, 0
```

The C programming Language III

- Designed to be compiled using a relatively straightforward compiler
- to provide low-level access to memory and language constructs that map efficiently to machine instructions.

- Designed to work cross-platform. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with **few change**s to its source code.
    - Java was designed with even better cross platform support under the tagline “Write once, run anywhere”.
- The language is available on various platforms, from embedded microcontrollers to supercomputers.

The C programming Language IV

Listing 2: Hello World in Standards Compliant C

```
#include <stdio.h>
int main(void)
{
   printf("hello, world\n");
}
```

Listing 2: Hello World in Standards Compliant python

```
def helloworld():
  print("Hello, world")

helloworld()
```

The C programming Language V

- “#include” is a pre-processing directive, it is saying to pull the contents of the specified file and replace this line with that “stdio.h” is part of Standard C.
- “main()” is a function, but it is a very special function
    - It acts as the entry point of the program it is from here that execution begins. Main returns an int to the calling environment.
- The next line calls (diverts execution to) a function named printf
    - This is a function found in the system library of C which sends output to the “standard out” of the calling environment, typically this means it prints out to the terminal or command prompt
    - That said std out can be redirected to funnel information between scripts or into files.
- printf will output the character array to the standard output.
- We do not have to explicitly return a value for main it implicitly returns “0” which means the program executed successfully.

The C programming Language VI

Write a C program to print your name, date of birth. and mobile number

```
#include <stdio.h>
int main(void)
{
  printf("Name: Dominic Carr\n");
  printf("DOB: June 12th, 1920\n");
  printf("086-1910000\n");
}
```

Write a python program to print your name, date of birth. and mobile number

```
def personinfo():
  print("Name: Dominic Carr")
  print("DOB: June 12th, 1920")
  print("086-1910000")
```

```
personinfo()
```

The C programming Language VII

Write a C program to print a block F using hash (#), where the F has a height of six characters and width of five and four characters

```
#include <stdio.h>
int main()
{
  printf("######\n");
  printf("#\n");
  printf("#\n");
  printf("#####\n");
  printf("#\n");
  printf("#\n");
  printf("#\n");
}
```

Python

```
def fpattern():
    print("######")
    print("#")
    print("#")
    print("#####")
    print("#")
    print("#")
    print("#")

fpattern()
```

Listing 3: Another Answer this time with a function

```
#include <stdio.h>
void print(int times, char a)
{
   for(int i = 0; i< times; i++)
   {
      printf("%c", a);
    }
    printf("\n");
}
int main()
{
   print(6,'#');
   print(1,'#');
   print(1,'#');
   print(5,'#');
   print(1,'#');
   print(1,'#');
}
```

python

```
def fpattern(n):
    for i in range(n):
        print("#", end='', flush=True)
    print(" ")
```

```
fpattern(5)
fpattern(1)
fpattern(1)
fpattern(5)
fpattern(1)
fpattern(1)
```

The C programming Language X

Write a C program to compute the sum of the two given integer values. If the two values are the same, then return triple their sum.

```
#include <stdio.h>
int sum(int a, int b)
{
   if (a==b)
   {
      return (a+b)*3;
    } else {
      return (a+b);
    }
}

int main()
{
   int res = sum(1,2);
   printf("Result is %d\n", res);
   res = sum(3,3);
   printf("Result is %d\n", res);
}
```

Python

```
def sumUp(a, b):
    if(a==b):
        return(a+b)*3
    else:
        return(a+b)

result = sumUp(1,2)
print("Result is", result)

result = sumUp(3,3)
print("Result is", result)
```

Structs I

Listing 4: C Example of representing a Person

```
#include <stdio.h>
struct person
{
  int age;
  float weight;
};

int main()
{
   struct person *personPtr, person1;
```

```
    personPtr = &person1;
    printf("Enter age: ");
    scanf("%d", &personPtr->age);
    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);
    printf("Age: %d\n", personPtr->age);
    return 0;

}
```

Structs III

- A struct is a way of grouping individual variables together
- It can be used to create a representation of something like a person
- As defined above a person has an age and a weight
- We will contrast this with OOP approaches which we will learn about soon

C vs Python

- C is compiled, Python is interpreted
- C allows low level memory access, Python does not
- Python support OOP, C does not
- Python has a much larger set of built-in functionality than C
- C code execution is much faster than Python
    - Compilation is key
- Variable types must be declared in C, not so in Python
- C has a more verbose syntax than Python
    - Python would be considered easier to learn
- In C memory management is manual, Python has automated management
- There are many syntactical differences, but some commonalities

Installing C on Windows

- Install Cygwin, which gives us a Unix-like environment running on Windows.
- Install a set of Cygwin packages required for building GCC.
- From within Cygwin, download the GCC source code, build and install it.
- Then you should be able to compile and run C programs.
- Follow these steps which are detailed extensively @ https://preshing.com/20141108/how-to-install-the-latest-gcc-on-windows/
- See the wiki help section for more information installing cygwin

Online C compiler

https://www.onlinegdb.com/online_c_compiler and Compile our C code online!

C Practice Questions I

We will take a look at some of the "elementary" questions @ https://adriann.github.io/programming_problems.html Other

1.c

```
#include <stdio.h>
```

```
int main()
{
    printf("Hello, World\n");
    return 0;
}
```

1.py

```
print("Hello World")
```

2.c

```
#include <stdio.h>

int main()
{
    char name[20];

    printf("What is your name?");

    fgets(name,20,stdin);

    printf("Hello %s", name);

    return 0;
}
```

2.py

```
name = raw_input("What is your name?")

print("Hello %s, How are you?" %(name))
```

3.c

```
#include <stdio.h>
#include <string.h>

int main()
{
    char* name;

    printf("What is your name?");

    gets(name);

    if ((strcmp(name,"Alice")==0) || (strcmp(name,"Bob")==0))
    {
        printf("Hello %s\n", name);
    } else
    {
        printf("Hello lowly peasant!\n");
    }
    return 0;
}
```

3.py

```
name = raw_input("What is your name?")

if (name=="Bob") or (name=="Alice"):
    print("Hello %s, How are you?" %(name))
else:
    print("Hello lowly peasant!")
```

4.c

```
#include <stdio.h>
#include <string.h>

int main()
{
    int n;
    int sum = 0;

    printf("Please enter a Number: ");
    scanf("%d", &n);

    for(int i = 1; i<=n; i++)
    {
        sum += i;
    }

    printf("The sum of 1 to %d, was %d\n", n, sum);

    return 0;
}
```

4.py

```
n = input("Enter a number: ")

sum = 0

for i in range(1,n+1):
    sum = sum + i

print("The sum of 1 to %d, was %d" % (n, sum))
```

5.c

```
#include <stdio.h>
#include <string.h>

int main()
{
    int n;
    int sum = 0;
    printf("Please enter a Number: ");
    scanf("%d", &n);

    for(int i = 1; i<=n; i++)
    {
        if (((i%3)==0) || ((i%5)==0))
        {
            sum += i;
        }
    }

    printf("The sum of 1 to %d, was %d\n", n, sum);

    return 0;
}
```

5.py

```
n = input("Enter a number: ")

sum = 0

for i in range(1,n+1):
    if ((i % 3)==0) or ((1 % 5)==0):
        print i
        sum = sum + i

print("The sum of 1 to %d, was %d" % (n, sum))
```

6.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int n, sum = 0;
    int product = 1;
    char* choice = (char*) malloc(10 * sizeof(char));

    printf("Please enter a Number: ");
    scanf("%d", &n);
    printf("Do you want to do product or sum?");
    scanf("%s", choice);

    if (strcmp(choice, "product")==0)
    {
        for(int i = 1; i<=n; i++)
        {
            product *= i;
        }

        printf("The product of 1 to %d, was %d\n", n, product);

    } else if (strcmp(choice, "sum")==0)
    {
        for(int i = 1; i<=n; i++)
        {
            sum += i;
        }

        printf("The sum of 1 to %d, was %d\n", n, sum);

    }
    return 0;
}
```

6.py

```
n = input("Enter a number: ")
kind = raw_input("Do you want to do sum or product? ")

sum = 0
product = 1

if (kind == "sum"):
    for i in range(1,n+1):
        sum = sum + i
    print("The sum of 1 to %d, was %d" % (n, sum))
elif (kind == "product"):
    for i in range(1,n+1):
        product = product * i
    print("The product of 1 to %d, was %d" % (n, product))
else:
    print("I did not understand your request")
```

C Practice Questions II

Questions:

1. Write a C program to get the absolute difference between n and 51. If n is greater than 51 return triple the absolute difference.
2. Write a C program to check two given integers, and return true if one of them is 30 or if their sum is 30.
3. Write a C program to compute the sum of the two given integers. If the sum is in the range 10..20 inclusive return 30
4. Write a C program that accept two integers and return true if either one is 5 or their sum or difference is 5.
5. Write a C program to check if y is greater than x, and z is greater than y from three given integers x,y,z
6. Write a C program to check if two or more non-negative given integers have the same rightmost digit.

Abs Diff

abs_diff.c

```
// Write a C program to get the absolute difference between n and 51.
// If n is greater than 51 return triple the absolute difference.

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
  char* choice = (char*) malloc(10 * sizeof(char));
  int n,val;
```

```c

  printf("Please enter a Number: ");
  scanf("%d", &n);
  printf("Number entered is : %d\n", n);
  if (n>51){
    val=(abs(n-51))*3;
  }
  else {
    val=(abs(n-51));
  }
  printf("result: %d\n",val);
  return val;
}
```

Sum of integers

two_integers.c

```c
// Write a C program to check two given integers, and return
// true if one of them is 30 or if their sum is 30

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

int main()
{
  char* choice = (char*) malloc(10 * sizeof(char));
  int n1,n2,val;


 printf("Please enter first Number: ");
    scanf("%d", &n1);
  printf("Please enter second Number: ");
    scanf("%d", &n2);

  printf("Numbers entered is : %d and %d \n", n1,n2);
  if (n1==30 || n2==30 ||n1+n2==30){
    val=true;
  }
  else {
    val=false;
  }
  printf("val is: %d\n", val);
  if(val==1){
    printf("Either values or sum of both equils 30, therefore....");
    printf("result: %s\n","TRUE");
  }
  else {
    printf("Neither values or the sum of both equils 30, therefore....");
    printf("result: %s\n","FALSE");
  }
  return val;
}
```

Sources

- https://www.computerhope.com/jargon/i/imp-programming.htm
- https://en.wikipedia.org/wiki/Abstraction_(computer_science)
- https://www.w3resource.com/c-programming-exercises/basic-algo/index.php

Week 4 - Shop in C

c code

shop.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct Product {
  char* name;
    double price;
};

struct ProductStock {
    struct Product product;
    int quantity;
};

struct Shop {
    double cash;
    struct ProductStock stock[20];
    int index;
};

struct Customer {
    char* name;
    double budget;
    struct ProductStock shoppingList[10];
    int index;
};

void printProduct(struct Product p)
{
    printf("PRODUCT NAME: %s \nPRODUCT PRICE: %.2f\n", p.name, p.price);
    printf("-------------\n");
}

void printCustomer(struct Customer c)
{
    printf("CUSTOMER NAME: %s \nCUSTOMER BUDGET: %.2f\n", c.name, c.budget);
    printf("-------------\n");
    for(int i = 0; i < c.index; i++)
    {
        printProduct(c.shoppingList[i].product);
        printf("%s ORDERS %d OF ABOVE PRODUCT\n", c.name, c.shoppingList[i].quantity);
        double cost = c.shoppingList[i].quantity *
c.shoppingList[i].product.price;
        printf("The cost to %s will be €%.2f\n", c.name, cost);
    }
}

struct Shop createAndStockShop()
{
    struct Shop shop = { 200 };
    FILE * fp;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;

    fp = fopen("stock.csv", "r");
    if (fp == NULL)
        exit(EXIT_FAILURE);

    while ((read = getline(&line, &len, fp)) != -1) {
        // printf("Retrieved line of length %zu:\n", read);
        // printf("%s IS A LINE", line);
        char *n = strtok(line, ",");
        char *p = strtok(NULL, ",");
        char *q = strtok(NULL, ",");
        // create variables from the strings to populate the structures
                int quantity = atoi(q);
        double price = atof(p);
                // create a variable to make a permanent copy of the string that is pointed to only in n
        char *name = malloc(sizeof(char) * 50);
        strcpy(name, n);
                // instanssiate the first structure to go into the second structure and insert the values in
        struct Product product = { name, price };
        struct ProductStock stockItem = { product, quantity };
                // as pointed out in the video, a lot happens in this line, see the detailed description below
        shop.stock[shop.index++] = stockItem;
        // printf("NAME OF PRODUCT %s PRICE %.2f QUANTITY %d\n", name, price, quantity);
    }

    return shop;
}

void printShop(struct Shop s)
{
    printf("Shop has %.2f in cash\n", s.cash);
    for (int i = 0; i < s.index; i++)
    {
        printProduct(s.stock[i].product);
        printf("The shop has %d of the above\n", s.stock[i].quantity);
    }
}

int main(void)
{
    // struct Customer dominic = { "Dominic", 100.0 };
    //
    // struct Product coke = { "Can Coke", 1.10 };
    // struct Product bread = { "Bread", 0.7 };
    // // printProduct(coke);
    //
```

```
100.     // struct ProductStock cokeStock = { coke, 20 };
101.     // struct ProductStock breadStock = { bread, 2 };
102.     //
103.     // dominic.shoppingList[dominic.index++] = cokeStock;
104.     // dominic.shoppingList[dominic.index++] = breadStock;
105.     //
106.     // printCustomer(dominic);
107.
108.     struct Shop shop = createAndStockShop();
109.     printShop(shop);
110.
111. // printf("The shop has %d of the product %s\n", cokeStock.quantity,
     cokeStock.product.name);
112.
113.     return 0;
114. }
```

This line of code on **line 75** is packed with functionality and important to understand in order to reproduce the functionality for the assignments.

`shop.stock[shop.index++] = stockItem;`

You need to refer to the Shop structure defined on line 15 in the c code sample above. This structure is then instantiated on line 49 and named `shop`. This structure `shop` contains another structure `ProducStock` named `stock` and an array of 20 of this is defined.

So in order to loop through this array the index is created that needs to be maintained and used to fill the right indexed positions.

So in summary `shop` contains three named values, `cash`, `stock` and `index`therefore lines 72 and 73 creates the new structured values that is required to populate into the `stock` variable.

So in the line `shop.stock[shop.index++] = stockItem;` we insert into `shop.stock` at indexed position `shop.index` the new value created in lines 72 and 73 `stockItem` and while doing this incrementing `shop.index` by one by appending the ++ in the line of code.

stock csv file

stock.csv

```
Coke Can, 1.10, 100
Bread, 0.7, 30
Spaghetti, 1.20, 100
Tomato Sauce, 0.80, 100
Big Bags, 2.50, 4
```

Week 6 - Object Oriented Programming

1. Goals of this Session

2. Object-Oriented

- What is it?
- Examples of OOP
- Contrasted with Procedural
- Message Passing
- Impurity in OOP
- Inheritance
- Polymorphism
- Interface
- Composition
- OOP Languages

Goals of this Session

- To understand....
  - What is Object Oriented porgramming?
  - How is it different?
  - How does OOP acheive abstraction, isolation, and reusability?
  - The basics of the Java programming language
  - Difference from Python
  - How to write an OOP Program in Java

Paradigms I

Recall

- The term **programming paradigm** is the style or way of **thinking about and approaching problems.**
- The chosen paradigm affects how the code is **written and structured.**
- It can heavily influence how **one thinks** about the problem being solved

Object-Oriented

Object-Oriented I

- Objects are representation entities e.g. Lists, Animals etc.
- An object has both state and functionality
- The class of an object can be thought of as it’s template
  - Can specify what sort of states the object can have
  - What functionality the object can perform
- For example a Person is a class of Object, and “Dominic” is an instance of that class
  - State could include name, age, occupation, gender etc.
  - Each person will vary in these states

Object-Oriented II

- Object-Oriented programming builds up libraries of reusable Objects(code)
- Some OOP languages are “pure”, eveything in the language is an object
    - Ruby is a pure OOP language
        - One of it’s design goals was to have “everything as an object” and to be more OOP than python
        - Though since 2.2 python has increased it’s purity http://www.python.org/download/releases/2.2/descrintro/
    - Java and C++ are impure OOP languages
    - As “primitive” data types are not objects in those languages

Object-Oriented III

Listing 1: Person Class in Java

```
public final class Person {
  private final String name;
  private final int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
   }
   public String toString() {
     return name + " " + info;
   }
   public static void main(String[] args) {
     Person a = new Person("Alice", 12);
     Person b = new Person("John", 34);
   }
}
```

Object-Oriented IV

Listing 2: Person Class in Ruby

```
class Person
    def initialize(name, age)
       @name = name
       @age = age
    end

    def name
    # labelname is only in scope within the name
        method
    labelname = "Fname:"
    return labelname + @name
    end
end
```

Object-Oriented V

- Different languages, same paradigm
- We see some syntactic differences
- Ruby is dynamically typed
- Java is strongly typed

- Java is more verbose
- The first method shown in each is a special one
    - Referred to commonly as the constructor
    - Creates new instances of objects

Object-Oriented VI

Listing 3: C Example of representing a Person

```c
#include <stdio.h>
struct person
{
   int age;
   float weight;
};
int main()
{
   struct person *personPtr, person1;
   personPtr = &person1;
   printf("Enter age: ");
   scanf("%d", &personPtr->age);
   printf("Enter weight: ");
   scanf("%f", &personPtr->weight);
   printf("Age: %d\n", personPtr->age);
   return 0;
}
```

Object-Oriented VII

- A struct is a grouping of variables under a single name
- It is a data-type itself, variables can be declared of it’s type
- A struct has state, but not functionality
- Modifications to the state of a struct would be done by methods outside of the struct
- This is a major difference when the complexity of the programs grow.

Object-Oriented VIII

Message Passing

- Message passing is the means by which objects communicate with each other
- Most typically realised as method calls
    - written before runtime
    - but this is not always the case, some languages have more flexibility
- The format is typically:

```
    receiver_object.method_name(argument1, argument2, ...)
```

- This can be understand as we are send a message to the receiver object to perform method name with our given input.

Object-Oriented IX

Message Passing

- For Example:

```
math_solver.add(12, 30)
```

- We shall discuss in future cases where the object need not explicitly define the exact message handler ahead of time.

Object-Oriented X

Java: Impure

- In Java Primitives are not objects.
    - This includes the data-types: int, char, float, and double.
- This was done for efficiency purposes.
- What is the consequence?
    - We cannot pass them messages, we cannot invoke a method onprimitives
        1. .toString()
- The above would not be valid Java, however in Ruby
    1. to_s()
- Is perfectly valid as "1" is an object.
- This is arguable nicer for the programmer as it brings consistency to the language

Object-Oriented XI

Java: Impure

- This impurity was "dealt with" somewhat with the introduction of object wrappers for the primitives
    - Such as "Double" which is an object version of "double".
    - When Java needs to treat them as objects "auto-boxing" is performed to convert them
- Such wrappers are a specail kind of object, they are **immutable objects**
    - When it has a value, the value cannot be changed.
    - Another example would be the **String** class in Java
- Immutability is sometimes a very useful property for certain objects

Object-Oriented XII

Inheritence

- Inheritance is a relation between two classes.
- Superclass is inherited from
- Subclass is the inheritor, it gets the properties of the superclass
- Inheritance promotes reusability

```
class Mammal
  def breathe
    puts "inhale and exhale"
  end
end

class Lion < Mammal
  def speak
```

```
    puts "Roar!!!"
  end
end
```

Object-Oriented XIII

- Many OOP languages will allow only single inheritence, where a sub-class inherits state and functionality from only one parent class
  - There are cases where multiple inheritence is supported through mixins
  - Also it is possible for an object to take on additional roles by implementing an interface or through “ducktyping”

Object-Oriented XIV

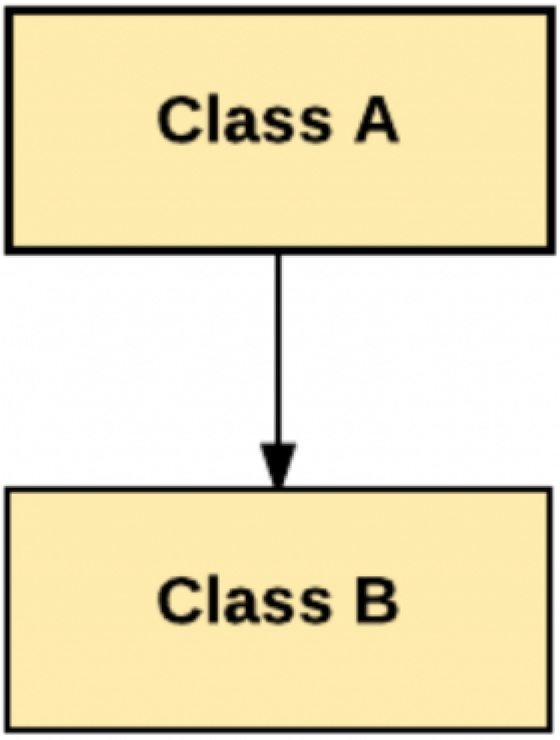


Figure: Single Inheritance

Object-Oriented XV

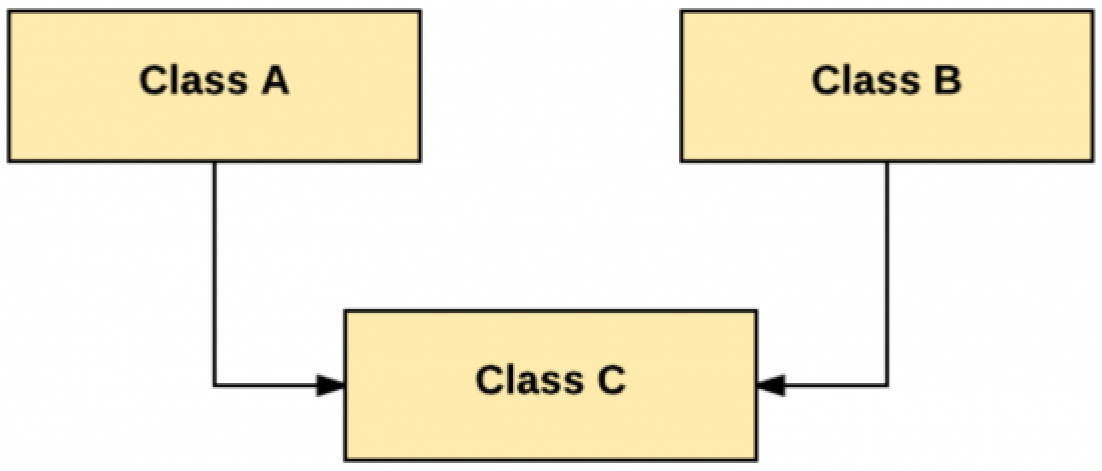


Figure: Multiple Inheritance

Object-Oriented XVI

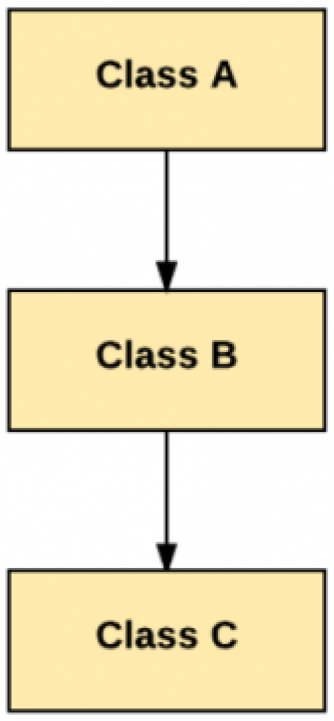


Figure: Multi-level Inheritance

- Class C inherits from B and it gets what B inherited from A
- In Ruby and Java (many others) all objects **implicitly** inherit from some base type.
    - In Java this is the class **Object**

Object-Oriented XVII

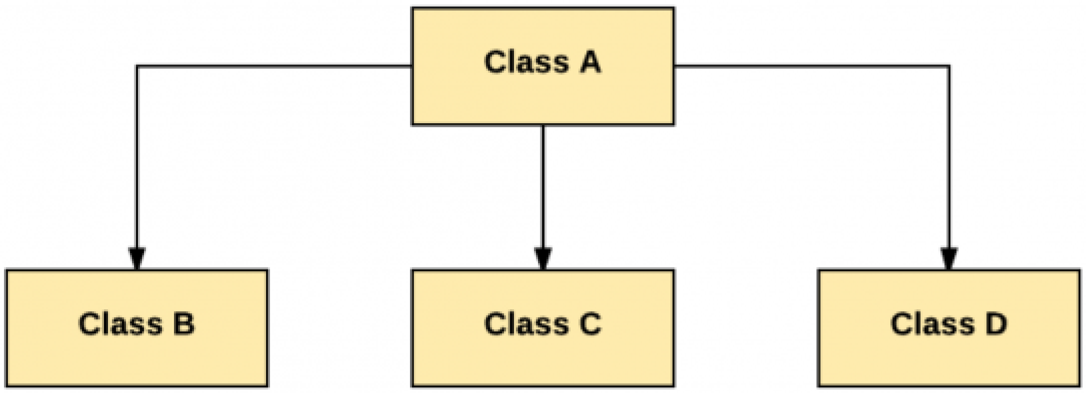


Figure: Hierarchical Inheritance

Object-Oriented XVIII

- The hierarchy is like what we had with the Mammal class

```
class Mammal
  def breathe
   puts "inhale and exhale"
  end
end

class Lion < Mammal
......

end

class Monkey < Mammal
......
end
```

- Both the Lion and the Monkey are Mammal and they both inherit from the Mammal class.

Object-Oriented XIX

- "Poly" means "many"
- Polymorphism is the ability of an object to take on many forms.
- The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
- For example a Monkey is a Mammal so when a reference is needed to a Mammal any Monkey can be substituted in.
- If a Lion was explicitly required then only it's sub-types could be subsituted.
- It is a very important concept and we will discuss it further we we take a deeper look at the Java language

Object-Oriented X

Interface

“an interface is a shared boundary across which two or more separate components of a computer system exchange information”

- Your mouse is an interface to the computer
- It moves the pointer around the screen
- But the same can be acheived with a track pad, or some kind of eye tracker etc.
- The point is the machine doesn’t care what is moving the pointer as long as the correct signals are sent from the input device (mouse,trackpad) to control the machine (OS).
- Developers regularly use Application Programming Interfaces (API) to communicate with webservices such as Twitter, Facebook, TVMaze,Weather etc.

Object-Oriented XXI

- A class which implements an interface does not gain any state from the interface, it merely takes on the requirement to implement the “contract” of the interface. It must do (or delegate) that which the interface promises to it’s users.
- It is possible for an object to inherit from a parent and implement a separate interface at the same time.
- We will see examples of this when we loom at Java and other languages.

Object-Oriented XXII

- An object can contain another object as an instance variable
  - Known as Object Composition
  - For example An Employee Object may have an Address object to represent their address
  - This stops the Employee class from being too complex as the Address class can deal with maintaining the address information and providing suitable methods for modifying.
- Object composition is used to represent “has-a” relationships
  - Every employee has an address
  - Every Student has many classes (So a Student object may have an Array of Module objects)

Object-Oriented XXIII

- OOP Languages
  - Python
  - Java
  - Ruby
  - PHP
  - C#
  - Objective C
  - Go
  - and many more!

Sources

https://www.computerhope.com/jargon/i/imp-programming.htm

https://en.wikipedia.org/wiki/Abstraction_(computer_science)

Java

Goals of this Session

- To Understand…
    - The Java Programming Language
    - The Java Virtual Machine
    - How to write programs in Java
    - How Inheritance works in Java

The Java Language

Java I

“Write once, run anywhere” - Java Language Tagline

Java II

- General-purpose programming language
- Class-based and object-oriented
    - Not a pure object-oriented language, as it contains primitive types
- Intended to let application developers write once, run anywhere(WORA)
    - compiled Java code can run on all platforms that support Java without the need for recompilation.
    - Java applications are typically compiled to bytecode that can run on
    - any Java virtual machine (JVM)
- Syntax similar to C and C++
- but it has fewer low-level facilities than either of them
    - For example Java does automatic memory management

Java III

Listing 1: Hello Java World!

```java
public class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!"); // Prints the string
to the console.
  }
}
```

Java IV

- So much to learn from this one snippet!
- the ".java" files must be named after the public class they contain so the above code must be in a file called "HelloWorldApp.java".
    - Don't forget this! Common source of errors!
- Java files are compiled into "bytecode", so when compiled this will result int "HelloWorldApp.class"
- TThe bytecode can be run by the JVM, we will talk in greater depth about the JVM later on.
- The keyword public denotes that a method can be called from code in other classes
- The keyword static in front of a method indicates a static method, which is associated only with the class and not with any specific instance of that class
    - Most methods we write will be instance methods and we will see the difference later on.

Java V

- The keyword void indicates that the main method does not return any value to the caller. If a Java program is to exit with an error code, it must call System.exit() explicitly.
    - This is quite similar to the main method in C, and there are many commonalities between C and Java syntactically.
    - Like in C the main method is the entry point into the program, though it may not be the first thing to execute.
- The main method accepts an array of strings as arguments to the program
    - this can be used to capture command line flags and other user input
- Printing is part of a Java standard library
    - The class System has a static variable "out" which represents the output stream, and you invoke the method println() on it to print your desired message to the screen
    - This is an OOP version of the printf function we seen in C.

Java VI

- Inheritance represents an "is-a" relationship between two classes for example all students are humans
    - so it makes sense to model students as a sub class of human
    - A student will have everything a human has but they will have some specific state of their own such as their grades!
- In Java the parent class is termed super class and the inherited class is the sub class
    - The keyword "extends" is used by the sub class to inherit the features of super class
    - Inheritance is important since it leads to reusability of code

```java
class subClass extends superClass
{
//methods and fields
}
```

Java VII

Listing 2: Person Object in Java

```java
public class Person {
   private int age;
   private String name;

   public Person(String name, int age){
      this.name = name;
      this.age = age;
   }

   public void setAge(int age){
      if (!(age <= 0 || age >= 110)){
      this.age = age;
      }
   }
```

Example I

```
public class Person {
   private int age;
   private String name;

   public Person(String name, int age){
      this.name = name;
      this.age = age;
   }

   public String toString(){
       return "Name:" + this.name + " Age: " + this.age;
   }

   public static void main(String[] args) {
       Person a = new Person("John", 23);
       Person b = new Person("Paul", 51);
       System.out.println(a);
       System.out.println(b);

   }


}
```

Java IX

How to run a java file on a mac in terminal?

Compiling and Running

```
$ javac Person.java

$ java Person
```

output:

# Name:John Age: 23

# Name:Paul Age: 51

- When the variables are private they cannot be accessed by callers
  - Modify the code to see what happens if you try to call any of the instance variables (Age and Name)
- We have access control on those variables and there is a method to modify the state of age
  - But it has custom logic so we cannot enter a silly value like “-1” for an age
  - OOP makes such “encapsulation” very easy.
- All objects in Java extend from Object, and from there we inherit some methods
  - toString is one! but why did I write it then?
  - We can override to provide our own implementation

Java X

Example II

Person.java

```
public class Person {
   private String name;
   private int age;

   public Person(String n, int a){
      this.name = n;
      this.age = a;
   }

   public String toString(){
       return "Name:" + this.name + " Age: " + this.age;
   }

   public void setAge(int n){
       if (n < 0){
           // this line is a comment, we want to finish method and not modify the age
           return;
       }
       this.age = n;

    }
}
```

PersonAppRunner.java

```
public class PersonAppRunner{

    public static void main(String[] args) {
        Person a = new Person("John", 23);
        Person b = new Person("Paul", 51);
        System.out.println(a);
        //a.age = -1;
        a.setAge(40);
        System.out.println(a);
        System.out.println(b);

    }
}
```

Student.java

```
//inheritance example
public class Student extends Person{

    private String[] classes;

    public Student(String n, int a, String[] c){
        super(n, a);
        this.classes = c;
    }

    public String toString(){
        String repr = super.toString() + "\nCLASSES: \n";
        for(int i=0; i<classes.length; i++){
            repr += classes[i] + "\n";
        }
        return repr;
    }

    public static void main(String[] args){
        String[] classes = new String[] {"Introduction to Maths", "Management for Computing", "Programming 1"};
        Student s = new Student("Pramod", 58, classes);
        s.setAge(59);
        System.out.println(s);
    }
    }
```

Java XI

- A student “is-a” Person
- So a Student has a name and age like a Person
    - We pass these up to the superclass (Person) and set up the new instance variable ourselves
- What will happen if we try to print out the student?
    - Lets Try!

Java XII

> > Paul is 25

* It looks like the student prints out just like a Person as it calls the superclass toString method

- Let’s modify it

-
  ```
  @Override
  public String toString(){
     String val = super.toString() + "\nTaking:\n";
     for(int i = 0; i<classes.length; i++){
        val += classes[i] + "\n";
     }
    return val;
   }
  ```

- We used the superclass method and added to it to print out the students classes
- Now the toString() is better suited

Java XIII

- but did we ever actually call toString?
    - Doesn’t look like it in the code right?
    - println method automatically calls toString()

Java XIV

- We were using an array in the Student class
- This is much like you would be used to in Python
- You can only add items of the same type
    - actually it is okay as long as they have a “is-a” relationship

Comparison with Python

Python OOP I

person.py

```
    class Person:
```

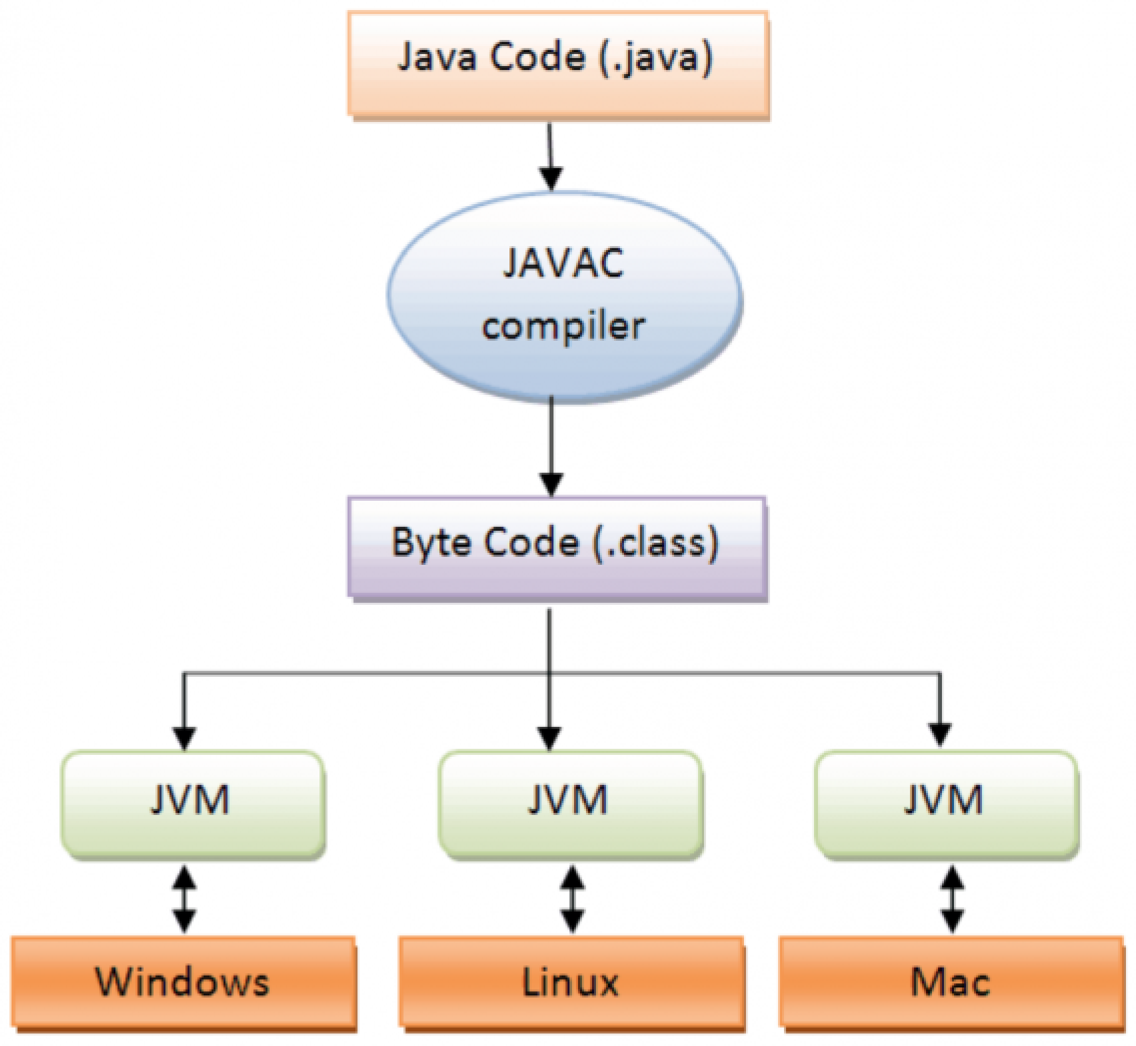
```
    def __init__(self, name, age):
      self.name = name
      self.age = age

  p1 = Person("John", 36)

  print(p1.name)
  print(p1.age)
```

Python OOP II

- Java is compiled, python is interpreted
- Java is more verbose than Python
- They both support OOP
- They have different syntax
- Java is statically typed, Python is dynamically typed
- Java is more rigid on structure, Python more flexible

Java Virtual Machine

JVM I



JVM II

- You write the code in Java
- The Java compiler produces bytecode
- The byte code is executed by the JVM
- Therefore Java code can run on any machine which supports the JVM
    - The same code can run on any OS supported by JVM
- This is a form of Abstraction
    - The JVM hides the differences between for example Windows, Linux, and MacOS.
    - With some caveats this works really well
- Java is the language in which most Android applications are written

Sources

https://www.computerhope.com/jargon/i/imp-programming.htm https://en.wikipedia.org/wiki/Abstraction_(computer_science) https://www.guru99.com/java-class-inheritance.html https://www.w3schools.com/python/python_classes.asp

Declarative Programming

- Goals of this session

- Declarative programming
    - What is it?
    - Example of Declarative
    - SQL
        - Standard SQL
        - Complex SQL
- Make
- Sources

Goals

- To understand
    - What is declarative programming?
    - How is it different from what we have seen?
    - Why is valuable?

Declarative programming

Programming by specifying the **result you want**, not how you get it.

```
SELECT UPPER(name)
FROM people
WHERE LENGTH(name) > 5
ORDER BY name
```

- Control flow is implicit: the programmer states only what the result should look like, not how to obtain it.
- No loops, no assignments, etc.
- Whatever engine that interprets this code is supposed to go get the desired information, and can use whatever approach it wants.
- SQL is declarative

Declarative II

```
SELECT * FROM Worker ORDER BY FIRST_NAME ASC;
```

- We would naturally assume the results will be pulled from a database table, but really the declaration of what we want is independent of how it is retrieved.
    - Could be from a text file, could be a human has to type out the response
    - The data store could be local or it could be on an AWS host in another country
    - The point of the declarative language is to abstract somewhat from the how.

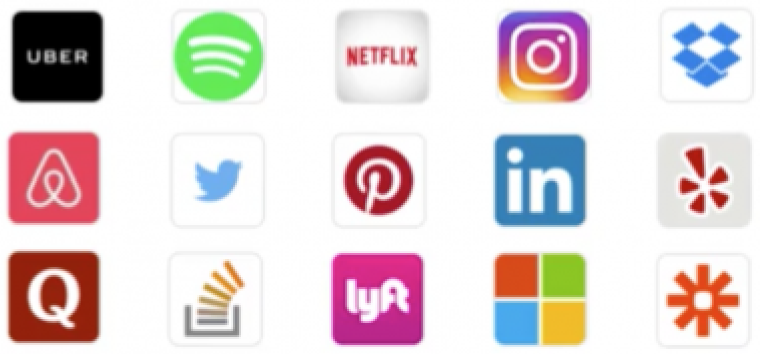
Declarative III

- Declarative programming
- contrasts with imperative and procedural programming
- Declarative programming is a non-imperative style of programming
    - Describe result not being explicit about how to obtain them.

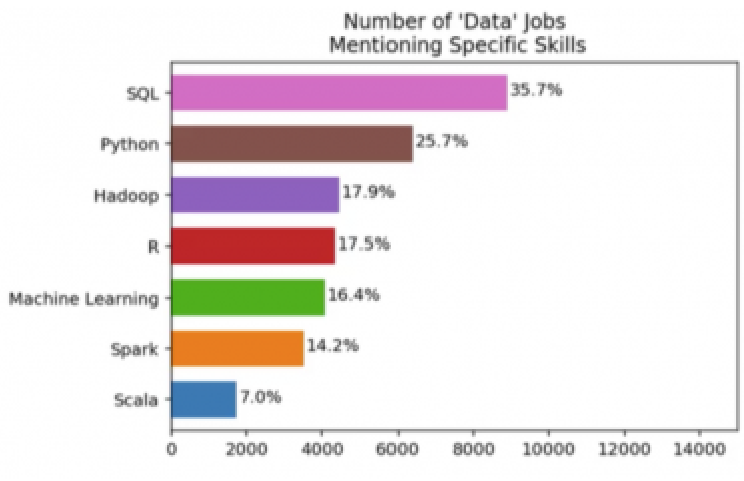
SQL

- Structured Query language
- Domain-specific language
  - Designed for managing data held in a relational database management
  - CRUD
- It is particularly useful in handing structured data, i.e data incorporating relations among entities and variables.
- Dates back to 1970
  - Edgar Codd wrote a paper describing a new system for organizing data in databases.
  - By the end 70s, prototypes of Codd's system has been built, and a query langugae - the **Structured Query Langugae(SQL)** - was born to interact with these databases.

Popularity of SQL


Companies using SQL

Popularity of SQL

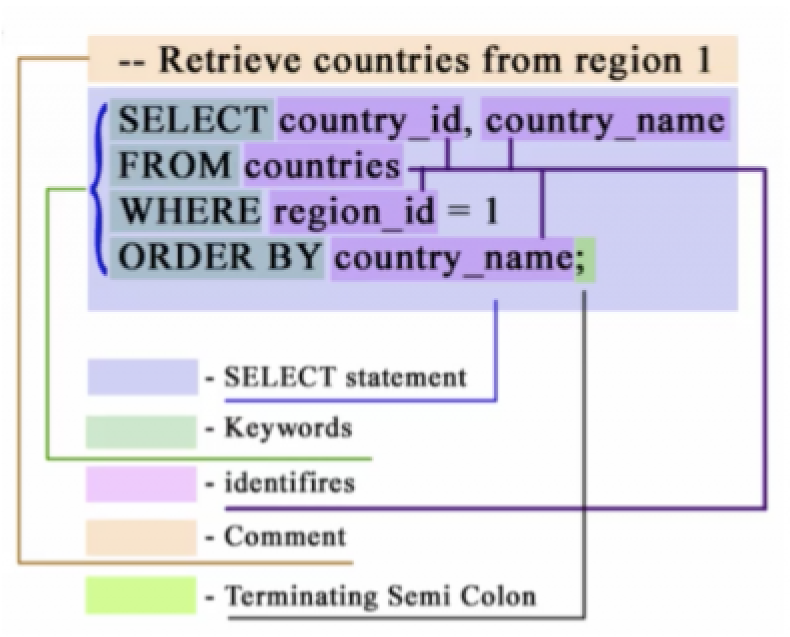


SQL Syntax

- The SQL language is subdivided into several language elements, including:
  - Clauses, which are constituent components of statements and queries.(In some cases, these are optional.)
  - Expressions, which can produce either scalar values, or tables consisting of columns and rows of data
  - Predicates, which specify conditions that can be evaluated to SQL three- valued logic(3VL)(true/flase/unknown) or Boolean truth values and are used to limit the effects of statements and queries, or to change program flow.

  - Queries, which **retrieve the data** based on specific criteria. This is an important element of SQL.
  - Statements, which may have a **persistent effect** on schemata and data or may control transactions, program flow, connections, sessions or diagnostics.

SQL Language Elements



Standard SQL

```
SELECT OrderID, Quantity,
CASE WHEN Quantity > 30 THEN "The quantity is greater than 30"
WHEN Quantity = 30 THEN "The quantity is 30"
ELSE "The quanity is under 30"
END AS QuantityText
FROM OrderDetails;
```

Comparison with C

Imagine the schema looks like:

```
personId INTEGER PRIMARY KEY, name VARCHAR(20), sex CHAR(1), birthday DATE, placeOfBrith
VARCHAR(20);
```

How would we find teh oldest person in the table?

```
SELECT * FROM people
 ORDER BYY birthday ASC
 LIMIT 1;
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
struct Person{
   char* name;
   char sex;
   int age;
   char* placeOfBirth;
}

int main(void){
   struct Person a = {"Anthony Stynes","M", 12, "New York"};
   struct Person b = {"Bertie Corr", "M", 1000, "Boston"};
   struct Person c = {"John Quin", "M", 33, "'Murica"};
}

struct Person arr[] = {a, b, c};
int maxAge = -1;
int index = 0;

for(int 1=0;i<3;i++){
   if(int i=0;i<3; i++){
       maxAge=arr[i].age;
       index=i;
    }
  }
printf("%s is oldest\n", arr[index].name);

return 0;
}
```

What's the most complex SQL query you ever wrote?

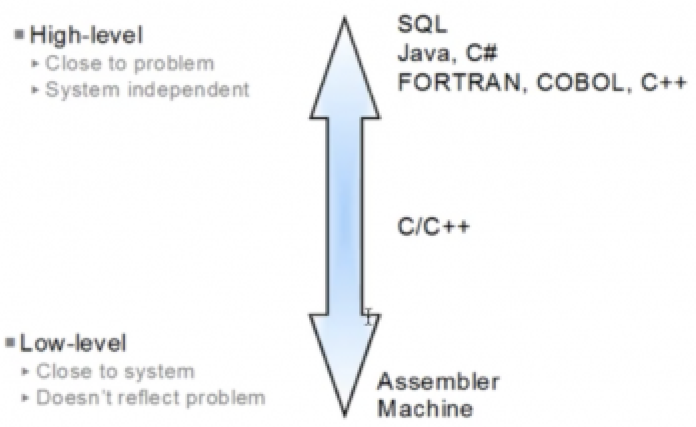
Greg Kemnitz, Postgres internals, embedded device db internals MySQL user level

*“I wrote a reporting query once that was **700 lines long** and visited **27 different tables** in lookups or joins. It didn't do any fancy stuff - just straigh whereclause predicates and joiing - but it was pretty gnarly and took **3 days to compose, debug and tune**”*

Bill Karwin, author of “SQL Antipatterns: Avoiding the pitfalls of database programming”

“One of the fun, complex sql queries I wrote was for a demo I did during a talk at OSCON 2006, SQL Outer Joins for Fun and Profit. It was a query that solved Sudoku puzzles”

Level of Abstraction



Make

- Make is a build automation tool
  - automatically builds executable programs and libraries from source code
  - ...by reading files called Makefiles which specify how to deriv the target program.
- Besides building programs, Make can be used to manage any project where some files must be updated automatically from others whenever the others change.

```
CFLAGS ?= -g

all:helloworld

helloworld: helloworld.o
   # Commands start with TAB not spaces
   $(CC) $(LDFLAGS) -o $@ $^

helloworld.o : helloworld.c
   $(CC) $(CFLAGS) -c -o $@ $<

clean: FRC
   rm -f helloworld helloworld.o

#This pseudo target causes all targets that depend on FRC
# to be remade even in case a file with the name of the target exists
# this works with any make implementation under the assumption that

# there is no file FRC in the current directory
FRC:
```

Functional Programming

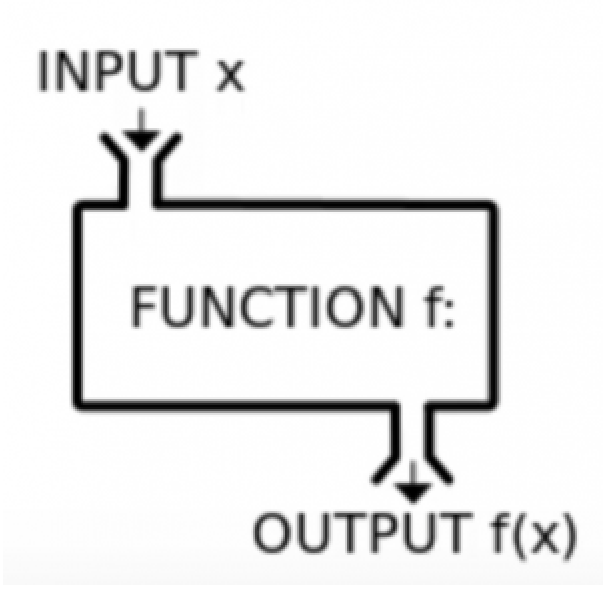
Functional programming is a subset of declarative programming.

Covered in this section:

Goals

- To understand
  - What is Functional programming?

Mathematical functions



$$f(x) = x^2$$

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ -(n+1)/2 & \text{if } n \text{ is odd} \end{cases}$$

Functional Programming

- Functional programming is a *programming paradigm*
  - A type of *declarative* programming
  - Sort of a sub-paradigm
- In FP computation is treated as the evaluation of mathematical functions
  - avoid changing-state and mutable data.
- In (pure) functional code, t**he output value of a function depends only on its arguments, so calling a function with the same value for an argument always produces the same result**.
- This is in **contrast to imperative programming** where, in addition to a function's arguments, **global program state can affect a function's resulting value**.
- Eliminating side effects, that is, changes in state, can make *understanding a program easier*

Functional Programming II

Functional programming languages are **declarative**, meaning that a computation's logic is expressed without d*escribing its control flow* Some languages *support functional programming* constructs without being purely functional.

Referential Transparency

Referential transparency is generally defined as the **fact** that an expression, in a program, may be replaced by its value (or anything having the same value) without changing the result of the program.

In other words we can replace any called to a **pure function** with it's result as the function result depends only on it's inputs.

```
def add(a, b):
   return a + b

def mult(a, b):
   return a * b

x = add(2, mult(3, 4))

print x
```

- any call to mult can be replaced by it's value so mult(3,4) can be replaced with it's result / value (12)
- mult has no side effects, it is a pure function. We couldn't safely do this with an impure function.

Referential Transparency I

```
call_counter = 0

def add(a, b):
  return a + b

def mult(a, b):
   global call_counter
   call_counter = call_counter + 1
   return a * b

x = add(2, mult(3, 4))

print x
```

- **In this version mult is impure**. It has side effects, as it changes the value of the call_counter global variable which persists after the function has returned.

Referential Transparency II

- Therefore we can no longer replace mult(3,4) with it's value, as the resulting program would have a different meaning, it would not be functionally equivialant.

Definition

- A pure function relies only it's inputs, and has no **side effects**.
- A call to a pure function will always return the same result.
- When a function performs any other "action", apart from calculating its return value, the function is impure
  - Most commonly this would be mutating state

Impure Functions I

- Impure functions can
  - Cause side effects
    - network calls
    - database calls
  - Not have the same result, with the same arguments, when called at different times
  - The return value of impure functions does not solely depend on its arguments
    - Hence, if you call the impure functions with the same set of arguments, you might get different return values.
  - If arguments are passed by reference an impure function might modify the arguments permanently.
    - Pointers in C
    - Objects in Java

Impure Functions II

```
class Person:
  def __init__(self, name, surname, age):
    self.name = name
    self.surname = surname
    self.age = age

  def can_vote(self, voting_age):
    if (self.age > voting_age):
       return True
    else:
       return False

person = Person("Jane",
                "Doe",
                 12)

print person.can_vote(18)

person.age = 19

print person.can_vote(18)
```

```
#Consider the following addition to the person class:


def same(self,p):
  if (p.name == self.name):
     p.name = "side effect"
     return True
   else:
     p.age = 101
     return False

#If we have
person = Person(
       "Jane",
       "Doe",
       12
)

john = Person(
       "John",
       "Joe",
        59
)

#and we perform

print person.same(john)

print john.age
```

are there side effects?

Consider this example:

```
limit = 100

def calculatebonus(num_sales):
   return 0.10 * num_sales if num_sales > limit else 0

def calculatebonus2(numSales):
   return 0.10 * num_sales if num_sales > 100 else 0

print(calculatebonus2(174))
```

Of the two functions, which is pure and which is not pure?

The answer is option one is impure because limit exists outside the function and affects the outcome in unpredictable ways.

Recursion

Definition

“Recursion in computer science is a method where t**he solution to a problem depends on solutions to smaller instances of the same problem** (as opposed to iteration). The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science”

Mostly this means that the programming language allows **a method to invoke itself**.

The Fibonacci Sequence I

Definition

“In mathematics, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones”

0, 1, 1, 2, 3, 5, 8, 13, 21, 34…

The Fibonacci Sequence II

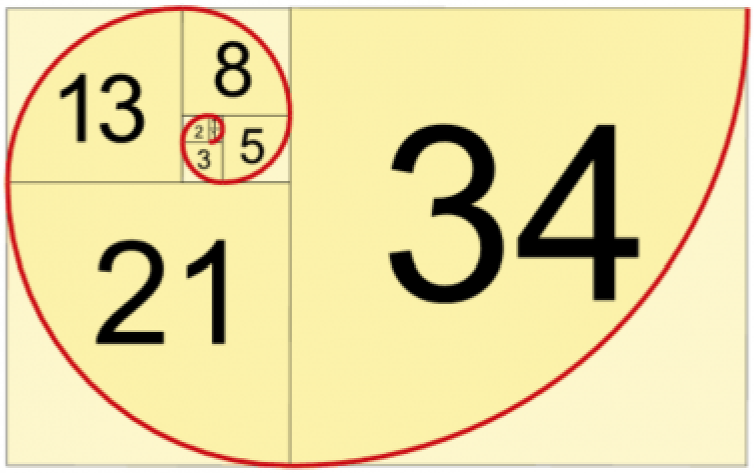


Figure: This is the Fibonacci Sequence

The Fibonacci Sequence III

Listing 1: Algorithm to determine the nth fibonacci number

```
def fib(n):
   if (n==1):
      return 0
   a = 0
   b = 1
   for i in range(2,n):
     c = a + b
     a = b
     b = c
   return b

for i in range(1,10):
   print fib(i)
```

This is an **iterative solution** to calculating the nth Fibonacci number, put simply this means that it is a method / function which loops to repeat some part of the code. A recursive method is one which calls itself to repeat the code.

The Fibonacci Sequence IV

This is an iterative solution to calculating the nth Fibonacci number, put simply this means that it is a method / function **which loops to repeat some part of the code**. A recursive method **is one which calls itself to repeat the code.**

Listing 2: Output of the program

```
0
1
1
2
```

```
3
5
8
13
21
```

The Fibonacci Sequence V

So do we achieve a recursive solution for this problem?

- From the definition of the Fibonacci sequence we know that fib(3) is fib(2) + fib(1)
- So maybe fib(n) calls fib(n-1) and fib(n-2)? let’s give it a go.

The Fibonacci Sequence VI

- That didn’t work!

File “1.py”, line 2, in fib

return fib(n-1) + fib(n-2)

RuntimeError: maximum recursion depth exceeded

- We don’t have anything to indicate that we should stop, so the algorithm runs until we run out of space.
- **“In computing, recursion termination is when certain conditions are met and a recursive algorithm stops calling itself and begins to return values“**

The Fibonacci Sequence VII

For the Fibonacci we can define three conditions:

- if n is 0, returns 0.
- if n is 1, returns 1.
- otherwise, return fib(n-1) + fib(n-2)

Listing 3: Recursive implementation of fib

```python
def fib(n):
   if (n==1 or n==0):
      return n
   return fib(n-1) + fib(n-2)

for i in range(1,10):
   print fib(i)
```

The Fibonacci Sequence VIII

What we have done in this specific case can be applied to all recursive algorithms. As all such algorithms must have the following properties to operate.

- Base Case (i.e. when to stop)
- Work toward Base Case

- Recursive Call

Factorial I

Definition

“In mathematics, the factorial of a non-negative integer n, denoted by n!,is the product of all positive integers less than or equal to n.”

5! = 5 × 4 × 3 × 2 × 1 = 120

Factorial II

Listing 4: Iterative calculation of the factorial of n

```
def factorial(n):
   factorial = 1;
   for i in range(2,n+1):
     factorial = factorial * i
    return factorial

print factorial(5)
```

How to do it recursively?

- 1! and 0! are both 1.
- that is a stop condition
- factorial(5) is 5 * factorial(4)

Factorial III

Listing 5: Recursive solution to factorial of N)

```
def factorial(n):
if (n == 0):
return 1
return n * factorial(n-1);
print factorial(5)
```

Why do we not need a condition for 1 as a special case?

Tasks I

Recursive solutions to the following:

- Calculate powers e.g. power(3,3), where that would be 3 * 3 * 3
- Determine if a word is a palindrome, this means the same backwards as forwards e.g. poop, madam…
- Reverse a String, i.e. given dominic return cinimod.

Calculate Powers

```
def pow(num, power):
    if (power == 1):
        return num
    return num * pow(num, power-1)

if __name__ == '__main__':
  v=pow(3,4)
  print('{}'.format(v))
```

Palindrome

```
def palindrome(word):
    length = len(word)
    #print("length was %d" % length)
    if (length <= 1):
        return True
    #print( "comparing %s with %s result is %r" % (word[0], word[length-1], (word[0] ==
word[length-1])))
    return (word[0] == word[length-1]) and palindrome(word[1:length-1])

if __name__ == '__main__':
  print('{}'.format(palindrome('madam')))
```

Reverse

```
def reverse(word):
    length = len(word)
    if (length <= 1):
        return word
    return word[length-1] + reverse(word[:length-1])

if __name__ == '__main__':
    print('{}'.format(reverse('cinimod')))
```

Iterable

This example in Python 2.7, the code below does not work for 3.7.

```
class StringIterator:
    def __init__(self, string):
        self.string = string
        self.index = 0
        self.length = len(string)

    def __iter__(self):
        return self

    def next(self):
        if self.index >= self.length:
            raise StopIteration
        else:
            self.index+=1
        return self.string[self.index-1]

print filter((lambda x: x != "i"), StringIterator("dominic"))
```

Lambdas and map

```
x = [1,2,3,4,5,6]

square = lambda x: x * x

print (square(2))

result = list(map(square, x))

print (result)

print (list(map(square, result)))
```

Filter and Reduce

```
import functools

print(functools.reduce((lambda x, y: x + y), [1,2,3,4,5,6]))
# 21

print(list(filter((lambda x: x > 3), [1,2,3,4,5,6])))
#[4, 5, 6]
```