**Big Array**

***Make an Array. Fill it with 19 random numbers from 20 to 90***

1. Print the Array from the beginning to the end
2. Print the Array from the beginning to the end using a for-each loop
3. What number is in the middle of the Array?
4. What is the average of the first, last and middle numbers?
5. Find the smallest and the largest number in the Array
6. Switch the largest with smallest number. Print the list
7. Create a new random from 1 to 10 and insert it in the middle slot. Print the numbers.
8. Add 10 to every number in the List. Print all.
9. Replace the 3rd element in the array with 5 and print the number that was ousted (only use one method to complete this.)
10. What numbers are in the 50s?
11. What numbers are multiples of 4?
12. Is there a 60 in the list?
13. Check to see if all of the elements from front to back are in the same order from back to front
14. How many numbers are greater than the average?
15. How many of the numbers are even?
16. Copy all of the elements of the array into a new array but in reverse order
17. Write a program to shift every element of an array circularly right. E.g.-INPUT : 1 2 3 4 5 OUTPUT : 5 1 2 3 4
18. Find the sum of all of the digits of all of the elements

**More below – Part II**

**Part II of Big Array**

Create a Cat class with the code given below (based on your programming language of choice).

**Data file:**  bigArrayList.dat

The input file begins with a number indicating the number of cats in the pound. The rest of the input will be the cat’s name, weight, age and cost for each cat. Weight and cost are doubles.

Create an Array and fill it with the cats from the file.

Sample output is as follows

**Name weight    age cost**

Inky 15.69   2 $35.79

Panda 14.3   6 $15.03

Rascal 21.1   21 $0.00

Blacky 13.99   3 $26.89

Taffy 24.5   10 $56.89

Toby 17.2   10 $37.57

1. Print out all the cats (there is no toString() available)
2. Print the name of the 3rd cat.
3. The last cat has gained 10 pounds. Update the weight on the object. Print the new weight.
4. The cat named Rascal died. Find that cat and remove it from the list.
5. A new kitten was brought in (Angel, 3.6, 1, 25.99).  Insert it into the 2nd cell.
6. A new geriatric cat was found (Gimpy, 14.3, 10,  29.99). Put him on the list.
7. Print the updated list with a for-each loop
8. Replace the 3rd cat with (Sugar, 23.6, 7, 33.25) put the removed cat at the end of the list.
9. Switch the 2nd and 4th cats.
10. Print the names of the cats on the list.
11. Remove all cats under $26. Print the costs of each cat remaining on the list.
12. All cats heavier than 15 pounds need to go on a diet <--  no for-each this time.

               Print the names of the cats being put on a diet.

**Final Results should be:**

1. All the cats:

Name weight    age cost

Inky 15.69   2 $35.79

Panda 14.3   6 $15.03

Rascal 21.1   21 $0.00

Blacky 13.99   3 $26.89

Taffy 24.5   10 $56.89

Toby 17.2   10 $37.57

2. The 3rd cat is named: Rascal

3. The updated weight is: 27.2

7. The updated list is:

Inky 15.69   2 $35.79

Angel 3.6   1 $25.99

Panda 14.3   6 $15.03

Blacky 13.99   3 $26.89

Taffy 24.5   10 $56.89

Toby 27.2   10 $37.57

Gimpy 14.3   10 $29.99

10. The current cat names are:

Inky Blacky Sugar Angel Taffy Toby Gimpy Panda

11. The cats costing $26 or more actually cost:

35.79  26.89  33.25  56.89  37.57  29.99

12. The cats on a diet are:

Inky  Sugar  Taffy  Toby

**Python:**

class Cat:

def \_\_init\_\_(self, name="", weight=0, age=0, cost=0):

self.name = name

self.weight = weight

self.age = age

self.cost = cost



**Java:**

/\*\*

\* Cat

\* Use with Cats with BigArrayLists

\*/

public class Cat {

private String myName;

private double myWeight;

private int myAge;

private double myCost;

public Cat() {

myName = "";

myWeight = 0;

myAge = 0;

myCost = 0;

}

public Cat(String name, double weight, int age, double cost) {

myName = name;

myWeight = weight;

myAge = age;

myCost = cost;

}

/\*\*

\* Methods

\*/

public void setWeight(double weight) { myWeight = weight; }

public void setCost(double cost) { myCost = cost; }

public void setAge(int age){ myAge = age; }

public void setName(String name) { myName = name; }

public int getAge() { return myAge; }

public double getCost() { return myCost; }

public double getWeight() { return myWeight; }

public String getName() { return myName; }

}



**C#:**

public class Cat {

public string Name { get; set; }

public double Weight { get; set; }

public int Age { get; set; }

public double Cost { get; set; }

public Cat(string name = "", double weight = 0,

int age = 0, double cost = 0) {

Name = name;

Weight = weight;

Age = age;

Cost = cost;

}

}



**C++:**

#include <string>

using namespace std;

class Cat {

public:

string name;

double weight;

int age;

double cost;

Cat() : name(""), weight(0), age(0), cost(0) {}

Cat(string name, double weight, int age, double cost) :

name(name), weight(weight), age(age), cost(cost) {}

};



**C:**

#include <string.h>

typedef struct {

char name[50];

double weight;

int age;

double cost;

} Cat;

void initializeCat(Cat\* cat, const char\* name, double weight, int age, double cost) {

strncpy(cat->name, name, sizeof(cat->name) - 1);

cat->weight = weight;

cat->age = age;

cat->cost = cost;

}



**Ruby:**

class Cat

attr\_accessor :name, :weight, :age, :cost

def initialize(name = "", weight = 0, age = 0, cost = 0)

@name = name

@weight = weight

@age = age

@cost = cost

end

end



**F#:**

type Cat(name:string, weight:double, age:int, cost:double) =

member val Name = name with get, set

member val Weight = weight with get, set

member val Age = age with get, set

member val Cost = cost with get, set



**JavaScript:**

class Cat {

constructor(name = '', weight = 0, age = 0, cost = 0) {

this.name = name;

this.weight = weight;

this.age = age;

this.cost = cost;

}

}



**Kotlin:**

class Cat(var name: String = "", var weight: Double = 0.0, var age: Int = 0, var cost: Double = 0.0)



**Golang:**

package main

type Cat struct {

Name string

Weight float64

Age int

Cost float64

}

func NewCat(name string, weight float64, age int, cost float64) \*Cat {

return &Cat{name, weight, age, cost}

}



**Scala:**

class Cat(var name: String = "", var weight: Double = 0, var age: Int = 0, var cost: Double = 0)



**Lua:**

Cat = {}

Cat.\_\_index = Cat

function Cat.new(name, weight, age, cost)

local self = setmetatable({}, Cat)

self.name = name or ""

self.weight = weight or 0

self.age = age or 0

self.cost = cost or 0

return self

end



**Swift:**

class Cat {

var name: String

var weight: Double

var age: Int

var cost: Double

init(name: String = "", weight: Double = 0, age: Int = 0, cost: Double = 0) {

self.name = name

self.weight = weight

self.age = age

self.cost = cost

}

}



**Dart:**

class Cat {

String name;

double weight;

int age;

double cost;

Cat({this.name = '', this.weight = 0, this.age = 0, this.cost = 0});

}



**Rust:**

pub struct Cat {

pub name: String,

pub weight: f64,

pub age: i32,

pub cost: f64,

}

impl Cat {

pub fn new(name: &str, weight: f64, age: i32, cost: f64) -> Cat {

Cat {

name: String::from(name),

weight,

age,

cost,

}

}

}



**Julia:**

mutable struct Cat

name::String

weight::Float64

age::Int

cost::Float64

end

function Cat(name::String="", weight::Float64=0.0, age::Int=0, cost::Float64=0.0)

return Cat(name, weight, age, cost)

end



**R:**

Cat <- setRefClass(

"Cat",

fields = list(

name = "character",

weight = "numeric",

age = "integer",

cost = "numeric"

),

methods = list(

initialize = function(name = "", weight = 0, age = 0, cost = 0) {

.self$name <- name

.self$weight <- weight

.self$age <- age

.self$cost <- cost

}

)

)



**Visual Basic:**

Public Class Cat

Public Property Name As String = ""

Public Property Weight As Double = 0

Public Property Age As Integer = 0

Public Property Cost As Double = 0

Public Sub New(Optional name As String = "", Optional weight As Double = 0, Optional age As Integer = 0, Optional cost As Double = 0)

Me.Name = name

Me.Weight = weight

Me.Age = age

Me.Cost = cost

End Sub

End Class



**Elixir:**

defmodule Cat do

defstruct name: "", weight: 0.0, age: 0, cost: 0.0

def new() do

%Cat{}

end

def new(name, weight, age, cost) do

%Cat{name: name, weight: weight, age: age, cost: cost}

end

def set\_weight(cat, weight), do: %{cat | weight: weight}

def set\_cost(cat, cost), do: %{cat | cost: cost}

def set\_age(cat, age), do: %{cat | age: age}

def set\_name(cat, name), do: %{cat | name: name}

def get\_age(cat), do: cat.age

def get\_cost(cat), do: cat.cost

def get\_weight(cat), do: cat.weight

def get\_name(cat), do: cat.name

end



**Perl:**

package Cat;

use strict;

use warnings;

sub new {

my ($class, %args) = @\_;

my $self = {

\_name => $args{name} || "",

\_weight => $args{weight} || 0,

\_age => $args{age} || 0,

\_cost => $args{cost} || 0,

};

bless $self, $class;

return $self;

}

sub set\_weight {

my ($self, $weight) = @\_;

$self->{\_weight} = $weight;

}

sub set\_cost {

my ($self, $cost) = @\_;

$self->{\_cost} = $cost;

}

sub set\_age {

my ($self, $age) = @\_;

$self->{\_age} = $age;

}

sub set\_name {

my ($self, $name) = @\_;

$self->{\_name} = $name;

}

sub get\_age {

my $self = shift;

return $self->{\_age};

}

sub get\_cost {

my $self = shift;

return $self->{\_cost};

}

sub get\_weight {

my $self = shift;

return $self->{\_weight};

}

sub get\_name {

my $self = shift;

return $self->{\_name};

}

1;

