1. In functional languages once you declare a variable/value, you cannot change it. You can only reassign the value by declaring it again.

In functional languages, for any given input paramaters, you would get the same output every time. i.e. the output does not depend on machine state, time, operating system etc.

Functional languages are able to optimise tail recursion. This makes recursion much more efficient and is used as oppose to loops.

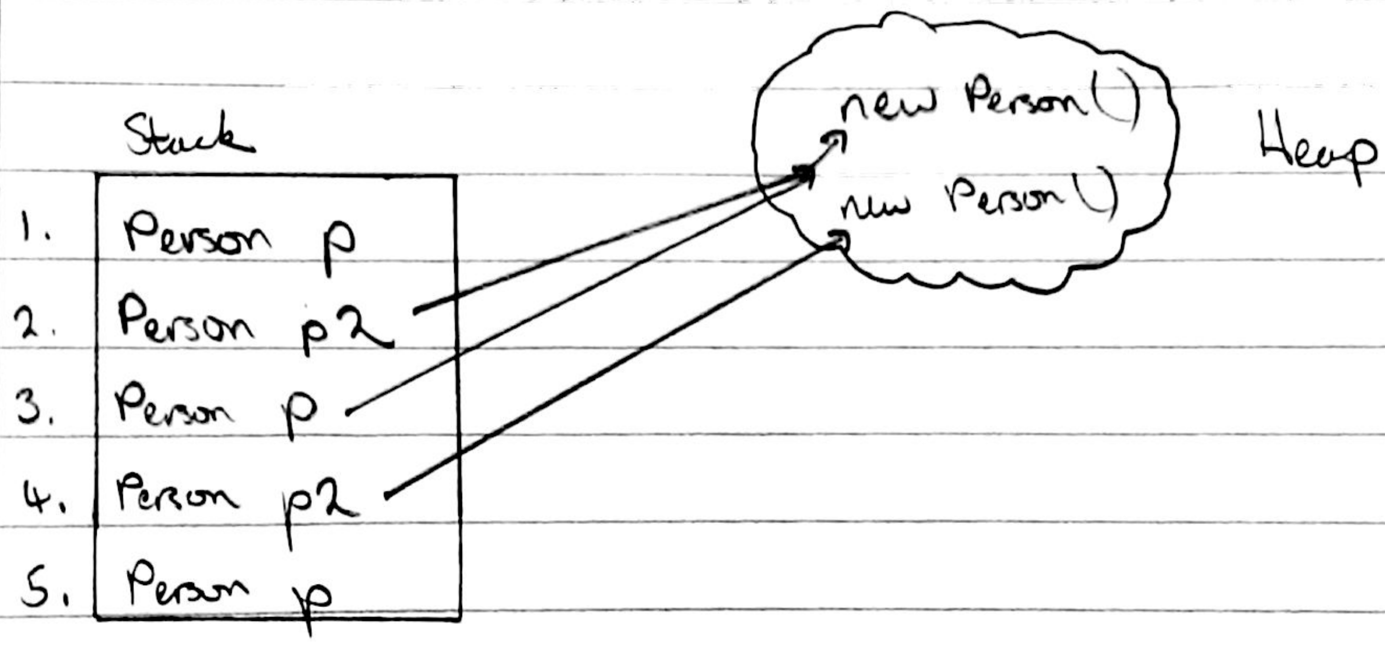
2.

```
// Primitives
int x = 5;

// References
Integer y;

// Classes
static class Table {
        int mWidth;
        int mHeight;
        Table(int width, int height){
                mWidth = width;
                mHeight = height;
        }
        void printSizes(){
                System.out.println(mWidth + "x" + mHeight);
        }
}
// Objects
Table myTable = new Table(10, 15);
```

3.



Stack
1.
Person p
2.
Person p2
3.
Person p
4.
Person p2
5.
Person p
new Person()
new Person()
Heap

4.

```
static int sum(int[] a){
	int s = 0;
	for(int i=0; i<a.length; i++)
		s += a[i];
	return s;
}

static int[] cumsum(int[] a){
	int[] cs = new int[a.length];
	cs[0] = a[0];
	for(int i=1; i<a.length; i++)
		cs[i] = cs[i-1] + a[i];
	return cs;
}

// Very dirty! Cannot figure out a better way to do this.
// Normally I would use an ArrayList.
static int[] positives(int[] a){
	int c=0;
	for(int i=0; i<a.length; i++)
		if(a[i] >= 0)
			c++;
	int[] p = new int[c];
	int j = 0;
	for(int i=0; i<a.length; i++){
		if(a[i] >= 0){
			p[j] = a[i];
			j++;
		}
	}
	return p;
}
```

5.

```
static float[][] nmatrix(int n){
	return new float[n][n];
}
```

6. This doesn't work because in vectorAdd, x and y are local variables in that frame. So when they are set, they are completely separate to a and b in main.

```
public static void main(String[] args) {
        int a=0;
        int b=2;
        int[] values = vectorAdd(a,b,1,1);
        a = values[0];
        b = values[1];
        System.out.println(a+" "+b);
}

static int[] vectorAdd(int x, int y, int dx, int dy){
        return new int[]{x+dx, y+dy};
}
```

7.

```
class LinkList{
        private int head;
        private LinkList tail;

        LinkList(int h, LinkList t){
                this.head = h;
                this.tail = t;
        }

        int count(){
                int i = 1;
                for(LinkList next = tail; next != null; next = next.getTail())
                        i++;
                return i;
        }

        int getNth(int n){
                LinkList next = this;
                for(int i=0; i<n; i++)
                        next = next.getTail();
                return next.getHead();
        }
```

```
int indexOf(int x){
	int i = 0;
	for(LinkList next = this; next.getHead() != x; next = next.getTail())
		i++;
	return i;
}

int indexOfObj(LinkList x){
	int i = 0;
	for(LinkList next = this; next != x; next = next.getTail())
		i++;
	return i;
}

int getHead(){
	return head;
}

LinkList getTail(){
	return tail;
}

void setTail(LinkList t){
	tail = t;
}

LinkList cons(int x){
	return new LinkList(x, this);
}

void dropLast(){
	LinkList next = this;
	while(next.getTail().getTail() != null)
		next = next.getTail();
	next.setTail(null);
}

void dropFirst(){
	this.head = tail.getHead();
	this.tail = tail.getTail();
}
```

```
	boolean isTailNull(){
		return tail == null;
	}

	public String toString(){
		String res = "";
		LinkList next = this;
		while(next != null){
			res += next.getHead() + ", ";
			next = next.getTail();
		}
		return res.substring(0, res.length() - 2);
	}
}
```

8.

```
	boolean testCycle(){
		int i = 0;
		LinkList next = this;
		while(next.getTail() != null){
			int index = this.indexOfObj(next);
			if(index < i) return true;
			next = next.getTail();
			i++;
		}
		return false;
	}
```

9. The stack holds a list of elements and allows you to add to the top of the stack with push, and remove the top of the stack with pop. Analogous to a stack of dishes.

10.
```
public class Stack {
        private LinkList stack = null;
        void push(int x){
                if(empty())
                        stack = new LinkList(x, null);
                else
                        stack = stack.cons(x);
        }
        void pop(){
                if(!empty())
                        stack.dropFirst();
        }
        int peep(){
                return stack.getHead();
        }
        int search(int x){
                return stack.indexOf(x) + 1;
        }
        public String toString(){
                return empty() ? "empty" : stack.toString();
        }
        boolean empty(){
                return stack == null;
        }
}
```