Part 1

Question 1: Prototypal Inheritance Exercises

- **Working with prototype**

  Ans. a) true (from rabbit)

  b) null (from animal)

  c) undefined (because jumps function is deleted from the parent object)

- **Searching algorithm**

Ans. a)

```
let head = {

            glasses: 1
};

let table = {
            pen: 3,
            __proto__: head
};

let bed = {
            sheet: 1,
            pillow: 2,
            __proto__: table
};

let pockets = {
            money: 2000,
            __proto__: bed
}
```

There is no difference at all while accessing glasses either from head or pockets.

- **Where does it write?**

Ans. It writes inside rabbit object. This is because this keyword is invoked from rabbit object. The function is looked up from animal but this (rabbit) is invoked inside rabbit.

- **Why are both hamsters full?**

Ans. This is because they both have a common stomach in the hamster object. To prevent this we can make separate stomach for two different types of hamster,

```
let hamster = {
                    stomach: [],
                    eat(food) {
                    this.stomach.push(food);
           }
          };

let speedy = {
  __proto__: hamster,
  stomach: []
};

let lazy = {
  __proto__: hamster,
  stomach: []
};
```

Question 2: F.prototype

- **Changing "prototype"**
  - 1)Ans. true. This is because assignment of prototype does not affect the existing ones. Here, prototype assignment is done after creating an object.
  - 2) Ans. false. This is because objects are based on reference. So when the prototype of main object is changed the values in their inheriting objects also change
  - 3) Ans. true. This is because the delete operation tries to delete a property in rabbits which does not exist and the eats is called from the parent object
  - 4) Ans. undefined. The property eats gets deleted from the parent object leaving it undefined

- **Creating an object with the same constructor**
  - **Ans. Correct Example**
    ```
    function Employee(name) {
              this.name = name;
    }

    let employee = new Employee ('John');
    let user2 = new employee.constructor('Pete');
    ```
  - **Ans. Incorrect Example**

```
function Employee(name) {
        this.name = name;
}
Employee.prototype = {};
let employee = new Employee ('John');
let user2 = new employee.constructor('Pete');
```

This is incorrect because we set the prototype of employee to be empty. When we invoke constructor method from user2 it tries to fine name constructor inside it and when it doesn't it moves up the hierarchy. Since, the constructor is removes from Employee it reaches to Object and creates a constructor from Object which is not what we want. And since there is no user2.name in Object we get a result undefined.

Question 3: Native Prototypes

- **Add method f.defer(ms) to functions**

```
Function.prototype.defer = function(ms) {
  setTimeout(this, ms);
};

function f() {
  alert("Hello!");
}

f.defer(1000);
```

- Add the decorating "defer()" to the function

```
Function.prototype.defer = function(ms) {
        let f = this;
        return function(...args) {
                setTimeout(() => f.apply(this, args), ms);
                }
};

function f(a, b) {
        alert( a + b );
}

f.defer(1000)(1, 2);
```