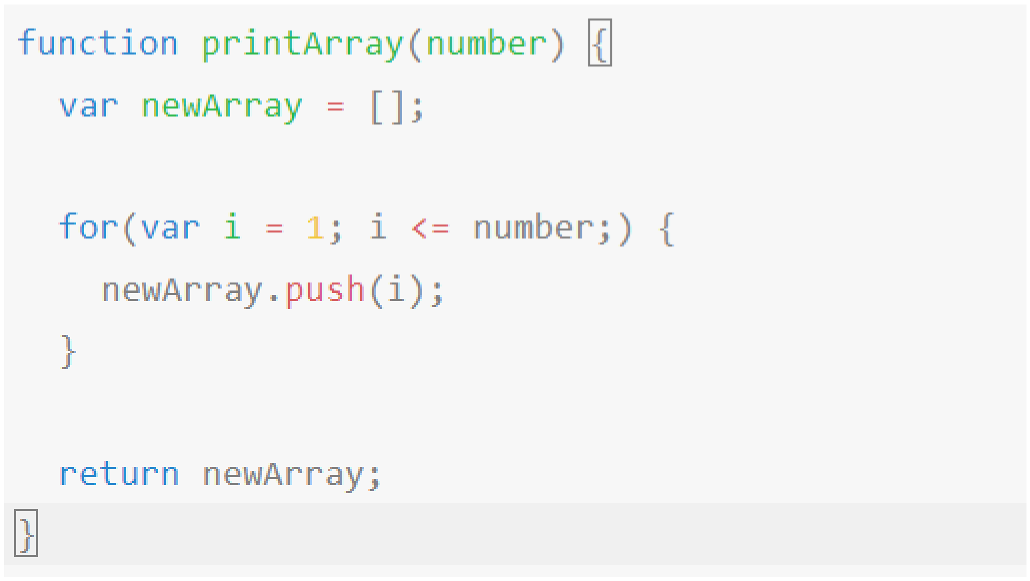
Buggy Code (Correct below code)

Manish created an infinite loop! Help him by fixing the code in the code tab to pass this challenge. Look at the examples below to get an idea of what the function should do.



```
function printArray(number) {
  var newArray = [];

  for(var i = 1; i <= number;) {
    newArray.push(i);
  }

  return newArray;
}
```

Examples

`printArray(1) ➞ [1]`

`printArray(3) ➞ [1, 2, 3]`

`printArray(6) ➞ [1, 2, 3, 4, 5, 6]`

Notes

- READ EVERY WORD CAREFULLY, CHARACTER BY CHARACTER!

How Much is True?

Create a function which returns the number of `true` values there are in an array.

Examples

```
countTrue([true, false, false, true, false]) ➞ 2

countTrue([false, false, false, false]) ➞ 0

countTrue([]) ➞ 0
```

Converting Objects to Arrays

Write a function that converts an object into an array, where each element represents a key-value pair in the form of an array.

Examples

```
toArray({ a: 1, b: 2 }) ➞ [["a", 1], ["b", 2]]

toArray({ shrimp: 15, tots: 12 }) ➞ [["shrimp", 15], ["tots", 12]]

toArray({}) ➞ []
```

Upvotes vs Downvotes

Given an object containing counts of both upvotes and downvotes, return what vote count should be displayed. This is calculated by subtracting the number of downvotes from upvotes.

Examples

```
getVoteCount({ upvotes: 13, downvotes: 0 }) ➞ 13

getVoteCount({ upvotes: 2, downvotes: 33 }) ➞ -31

getVoteCount({ upvotes: 132, downvotes: 132 }) ➞ 0
```

Older Than Me

Create a *method* in the `Person` class which returns how *another person*'s age compares. Given the instances `p1`, `p2` and `p3`, which will be initialized with the attributes `name` and `age`, return a sentence in the following format:

{other person name} is {older than / younger than / the same age as} me.

Examples

```
p1 = Person("Samuel", 24)
p2 = Person("Joel", 36)
p3 = Person("Lily", 24)
p1.compareAge(p2) ➞ "Joel is older than me."

p2.compareAge(p1) ➞ "Samuel is younger than me."

p1.compareAge(p3) ➞ "Lily is the same age as me."
```

Calculate the Total Price of Groceries

Create a function that takes an array of objects (groceries) which calculates the total price and returns it as a number. A grocery object has a product, a quantity and a price, for example:

```
{
  "product": "Milk",
  "quantity": 1,
  "price": 1.50
}
```

Examples

```
// 1 bottle of milk:
getTotalPrice([
  { product: "Milk", quantity: 1, price: 1.50 }
]) ➞ 1.5

// 1 bottle of milk & 1 box of cereals:
getTotalPrice([
  { product: "Milk", quantity: 1, price: 1.50 },
  { product: "Cereals", quantity: 1, price: 2.50 }
]) ➞ 4

// 3 bottles of milk:
getTotalPrice([
  { product: "Milk", quantity: 3, price: 1.50 }
]) ➞ 4.5

// Several groceries:
getTotalPrice([
  { product: "Milk", quantity: 1, price: 1.50 },
  { product: "Eggs", quantity: 12, price: 0.10 },
  { product: "Bread", quantity: 2, price: 1.60 },
  { product: "Cheese", quantity: 1, price: 4.50 }
]) ➞ 10.4

// Some cheap candy:
getTotalPrice([
  { product: "Chocolate", quantity: 1, price: 0.10 },
  { product: "Lollipop", quantity: 1, price: 0.20 }
]) ➞ 0.3
```

Notes

There might be a floating point precision problem in here...

Weekly Salary

Write a function that takes a list of `hours` and returns the total weekly salary.

- The input list `hours` are listed sequentially, ordered from Monday to Sunday.
- A worker earns $10 an hour for the first 8 hours.
- For every overtime hour, he earns $15.
- On weekends, the employer pays double the usual rate, *regardless how many hours were worked previously that week*. For instance, 10 hours worked on a weekday would pay 80+30 = $110, but on a weekend it would pay 160+60 = $220.

Examples

```
weeklySalary([8, 8, 8, 8, 8, 0, 0]) ➞ 400

weeklySalary([10, 10, 10, 0, 8, 0, 0]) ➞ 410

weeklySalary([0, 0, 0, 0, 0, 12, 0]) ➞ 280
```

Get Students with Best Test Avg.

Given an object with students and the grades that they made on the tests that they took, determine which student has the best Test Average. The `key` will be the student's name and the `value` will be an array of their grades. You will only have to **return the student's name**. You do not need to return their Test Average.

Examples

```
getBestStudent({
  John: [100, 90, 80],
  Bob: [100, 70, 80]
}) ➞ "John"

// John's avg = 90
// Bob's avg = 83.33

getBestStudent({
  Susan: [67, 84, 75, 63],
  Mike: [87, 98, 64, 71],
  Jim: [90, 58, 73, 86]
}) ➞ "Mike"
```

Notes

All students in an object will have the same amount of test scores. So no student will have taken more tests than another.

Shiritori Game

This challenge is an English twist on the Japanese word game **Shiritori**. The basic premise is to follow two rules:

1. **First character** of **next word** must match **last character** of **previous word**.
2. The word must not have already been said.

Below is an example of a **Shiritori** game:

```
["word", "dowry", "yodel", "leader", "righteous", "serpent"]  // valid!

["motive", "beach"]  // invalid! - beach should start with "e"

["hive", "eh", "hive"]  // invalid! - "hive" has already been said
```

Write a **Shiritori class** that has **two instance properties**:

- **words**: an array of words already said.
- **game_over**: a boolean that is true if the game is over.

and **two instance methods**:

- **play**: a method that takes in a word as an argument and checks if it is valid (the word should follow rules #1 and #2 above).
  - If it is valid, it adds the word to the **words** array, and returns the **words** array.
  - If it is invalid (either rule is broken), it returns **"game over"** and sets the **game_over** boolean to **true**.
- **restart**: a method that sets the **words** array to an empty one `[]` and sets the **game_over** boolean to **false**. It should return **"game restarted"**.

Examples

```
my_shiritori = Shiritori.new()

my_shiritori.play("apple") ➞ ["apple"]
my_shiritori.play("ear") ➞ ["apple", "ear"]
my_shiritori.play("rhino") ➞ ["apple", "ear", "rhino"]
my_shiritori.play("corn") ➞ "game over"
```

```
// Corn does not start with an "o".

my_shiritori.words ➞  ["apple", "ear", "rhino"]

// Words should be accessible.

my_shiritori.restart() ➞ "game restarted"
my_shiritori.words ➞ []

// Words array should be set back to empty.

my_shiritori.play("hostess") ➞ ["hostess"]
my_shiritori.play("stash") ➞ ["hostess", "stash"]
my_shiritori.play("hostess") ➞ "game over"

// Words cannot have already been said.
```

Notes

- The **play** method should **not** add an invalid word to the **words** array.
- You don't need to worry about capitalization or white spaces for the inputs for the **play** method.
- There will only be **single inputs** for the **play** method.
- Read more about Shiritori in the **Resources** tab.

Coffee Shop

Write a **class** called **CoffeeShop**, which has **three instance variables**:

1. **name** : a string (basically, of the shop)
2. **menu** : an array of items (of object type), with each item containing the **item** (name of the item), **type** (whether *food* or a *drink*) and **price**.
3. **orders** : an empty array

and **seven methods**:

1. **addOrder**: adds the **name** of the item to the end of the **orders** array if it exists on the **menu**. Otherwise, return `"This item is currently unavailable!"`
2. **fulfillOrder**: if the **orders** array is **not empty**, return `"The {item} is ready!"`. If the **orders** array is empty, return `"All orders have been fulfilled!"`
3. **listOrders**: returns the list of **orders** taken, otherwise, an **empty** array.
4. **dueAmount**: returns the total amount due for the **orders** taken.
5. **cheapestItem**: returns the **name** of the cheapest item on the menu.
6. **drinksOnly**: returns only the *item* **names** of *type* **drinks** from the menu.
7. **foodOnly**: returns only the *item* **names** of *type* **food** from the menu.

IMPORTANT: Orders are fulfilled in a **FIFO** (first-in, first-out) order.

Examples

```
tcs.addOrder("hot cocoa") ➞ "This item is currently unavailable!"
// Tesha's coffee shop does not sell hot cocoa
tcs.addOrder("iced tea") ➞ "This item is currently unavailable!"
// specifying the variant of "iced tea" will help the process

tcs.addOrder("cinnamon roll") ➞  "Order added!"
tcs.addOrder("iced coffee") ➞ "Order added!"
tcs.listOrders ➞ ["cinnamon roll", "iced coffee"]
// the list of all the items in the current order

tcs.dueAmount() ➞ 2.17

tcs.fulfillOrder() ➞ "The cinnamon roll is ready!"
tcs.fulfillOrder() ➞ "The iced coffee is ready!"
tcs.fulfillOrder() ➞ "All orders have been fulfilled!"
// all orders have been presumably served

tcs.listOrders() ➞ []
// an empty array is returned if all orders have been exhausted

tcs.dueAmount() ➞ 0.0
// no new orders taken, expect a zero payable

tcs.cheapestItem() ➞ "lemonade"
tcs.drinksOnly() ➞ ["orange juice", "lemonade", "cranberry juice",
"pineapple juice", "lemon iced tea", "vanilla chai latte", "hot
chocolate", "iced coffee"]
tcs.foodOnly() ➞ ["tuna sandwich", "ham and cheese sandwich", "bacon and
egg", "steak", "hamburger", "cinnamon roll"]
```

Notes

Round off **due amount** up to **two decimal** places.