# Memory management.
# Garbage Collector.
# Dev Tool

# Основні типи змінних в JS

- boolean
  - true or false
- number
  - any double precision IEEE 754 number
- string
  - UTF-16 string
- object
  - key value map

# Розмір об’єкта

Два розміра:

- Shallow
  - лише сам об’єкт
  - зазвичай не велика стала величина
- Retained
  - shallow + сума всіх залежних об’єктів

Shallow == Retained, для змінних типу string, number, boolean

# Навіщо це все потрібно

- Попередити появу “меморі ліків”(memory leaks)
- Забезпечити найкращу взаємодію з користувачем
- Девайс більше часу працює від зарядки
- Попередити “креш” додатка

# Memory leaks

Це поступова втрата доступної оперативної памяті компютера.

Коли програма систематично не повертає пам’ять, яку їй виділили в тимчасове використання

# Memory leaks

- Замикання
- Збережене посилання  на видалений DOM елемент
- Видалення DOM елемента на який було “повішено” слухача подій
- Timeout, interval

# Життєвий цикл пам’яті

Незалежно від мови програмування, життєвий цикл пам'яті практично завжди один і той же:

- Виділення необхідної пам'яті.
- Використвння виділеної пам'яті (читання, запис).
- Звільнення виділеної пам'яті, коли в ній більше немає необхідності.

# Виділення пам'яті в JavaScript

Виділення пам'яті при ініціалізації значень змінних.

```
var n = 123; // виділяє пам`ять для типу number
var s = "azerty"; // виділяє пам'ять для типу string
var o = {
  a: 1,
  b: null
}; // виділяє пам'ять для типу object та всіх його внутрішніх змінних
```

# Виділення пам'яті в JavaScript

```
var a = [1, null, "abra"]; // (like object) виділяє пам'ять для array та його внутрішніх значень
function f(a){
  return a + 2;
} // виділяє пам'ять для function (яка представляє собою об'єкт, що викликається)
// функціональні вирази також виділяють пам’ять під object
someElement.addEventListener('click', function(){
  someElement.style.backgroundColor = 'blue';
}, false);
```

# Виділення пам'яті в JavaScript

Деякі методи виділяють пам'ять для нових значень або об'єктів

```
var a = ["ouais ouais", "nan nan"];
var a2 = ["generation", "nan nan"];
var a3 = a.concat(a2); // новий масив з 4 елементами в результаті конкатенації масивів 'a' и 'a2'
```

# Використання значень

"Використання значень", як правило, означає - читання і запис значень з / в виділеної для них області пам'яті.

Це відбувається при читанні, записі значення якої-небудь змінної, або властивості об'єкта, також навіть при передачі аргументу функції.

# Звільнення пам'яті, коли вона більше не потрібна

Здається, це робота для нього )

# Garbage Collector

“Збирач сміття” стежить за виділенням та використанням пам'яті і при необхідності автоматично звільняє не потрібні більше ділянки пам`яті.

# Що таке сміття?

це та частина змінних які більше не будуть використовуватись

# Типи перадачі змінних

- По значенню
  - boolean
  - number
  - string
- По посиланню
  - object (oject, array, function)

# Стара та нова школи

- Підраховуємо посилання

- Використовуємо принцип досяжності

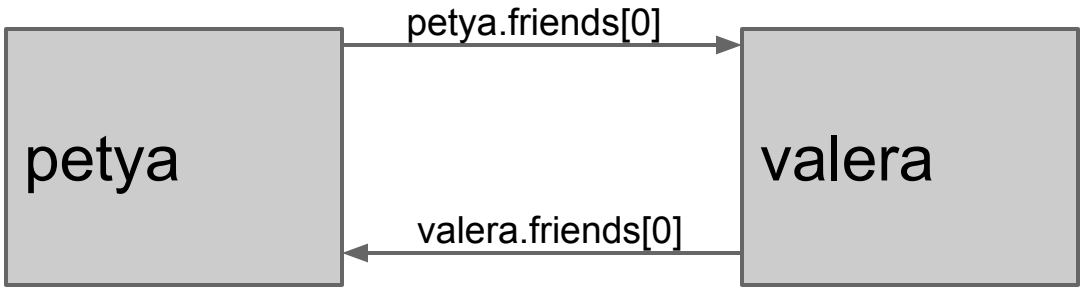
# Стара школа

- Переваги
  - доволі легко реалізовується
  - легко для розуміння
- Недоліки
  - не вирішує проблеми циклічних посилань
  - в зв’язку з попереднім легко з’являються “меморі ліки”

# Циклічні посилання

```
var petya = {
  friends: [valera]
}
```

```
var valera = {
  friends: [petya]
}
```



# Нова школа (принцип досяжності)

Певна множина значень спочатку вважається досяжною:

- Значення, посилання на які містяться в стеку виклику, тобто - всі локальні змінні і параметри функцій, які зараз виконуються або знаходяться в очікуванні закінчення вкладеного виклику.
- Всі глобальні змінні.

Ці значення гарантовано зберігаються в пам'яті. Ми будемо називати їх коренями.

# Принцип досяжності

Будь-яке значення зберігається в пам'яті лише до тих пір, поки воно досяжне з кореня за посиланням, або по ланцюжку посилань.

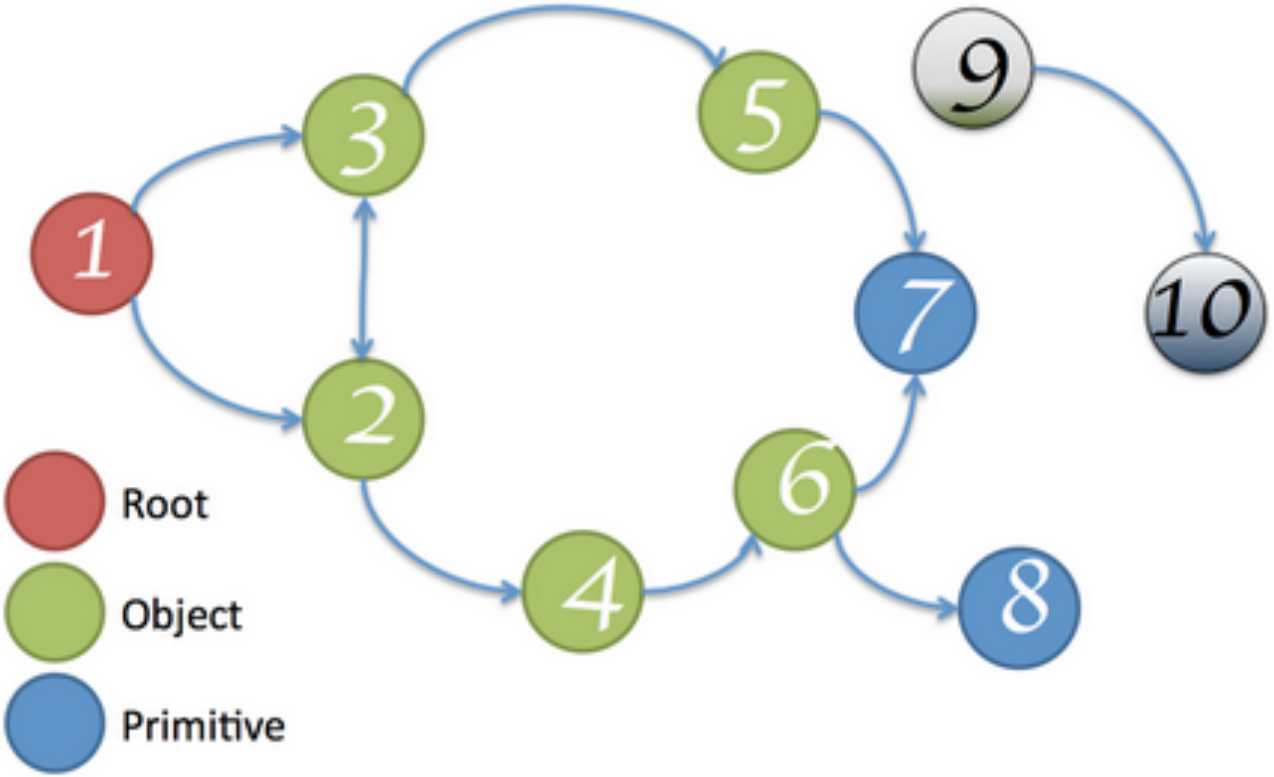
# Досяжність та наявність посилань

Простіше кажучи:

“значення залишається в пам'яті, поки на нього є посилання.”

- Якшо посилань на значення немає, то пам’ять, яку воно займало, вивільняється
- Наявність  посилання не гарантує, що значення залишиться в пам’яті

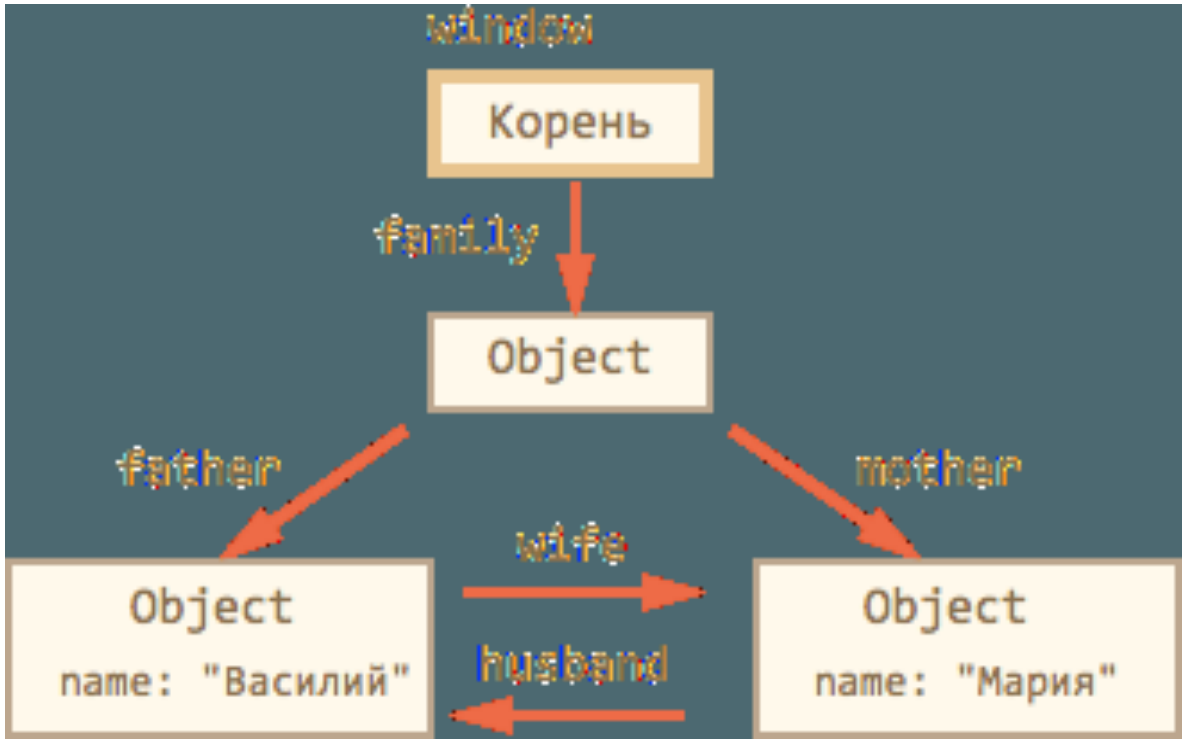
# Алгоритм збірки сміття



# Алгоритм збірки сміття

## Приклад:

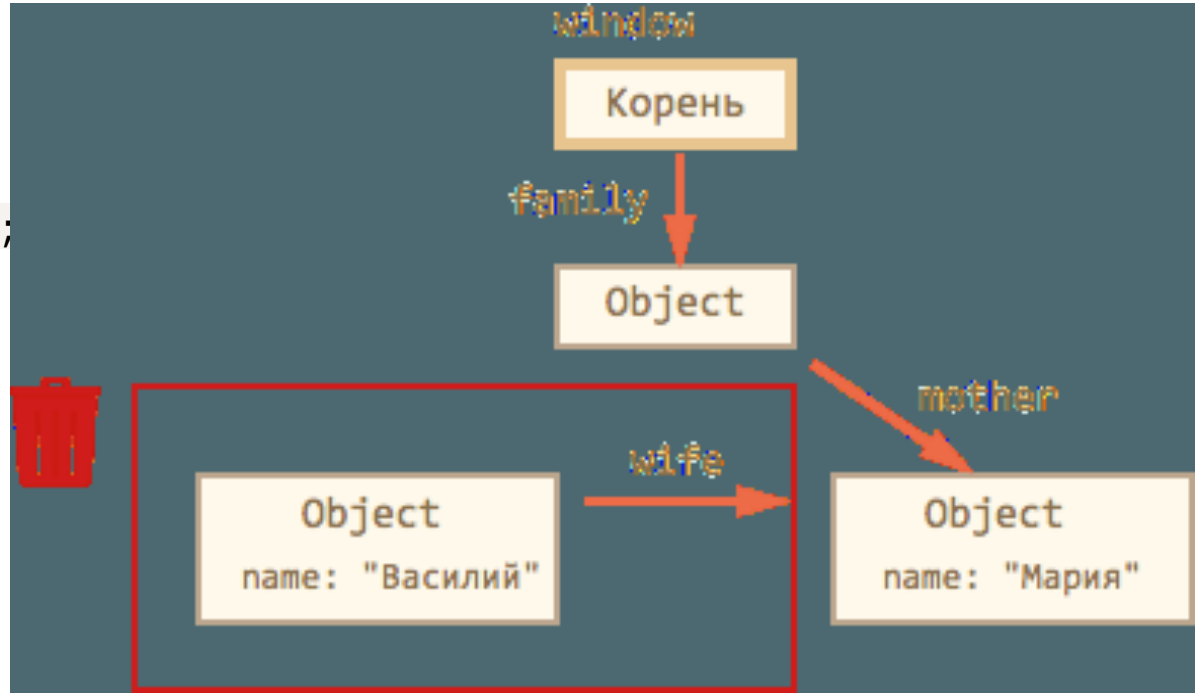
```
function marry(man, woman) {
  woman.husband = man;
  man.wife = woman;

  return {
    father: man,
    mother: woman
  }
}

var family = marry({
  name: "Василий"
}, {
  name: "Мария"
});
```



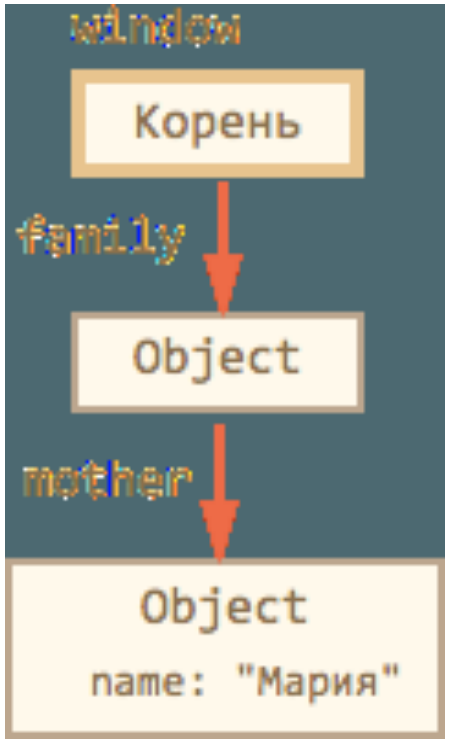
# Алгоритм збірки сміття

Руйнуємо сім’ю:

```
delete family.father;
delete family.mother.husband;
```



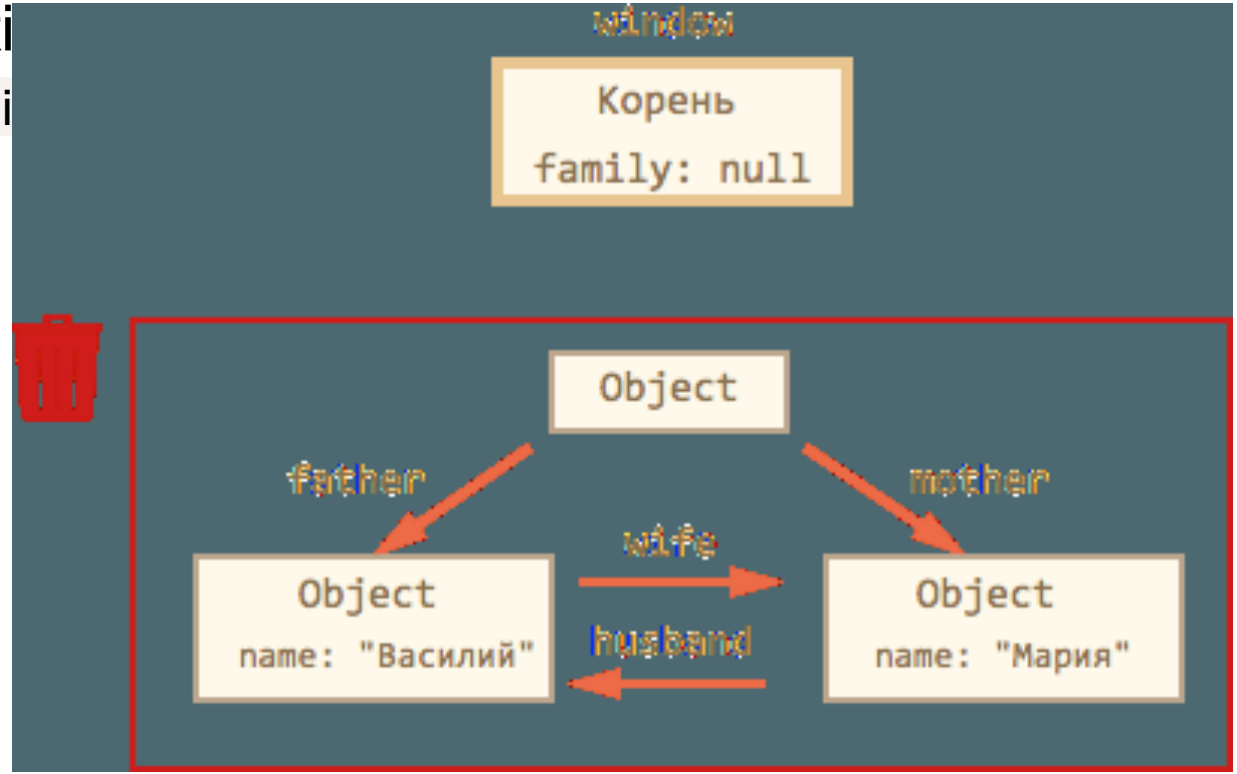
# Алгоритм збірки сміття



# Алгоритм збірки сміття

У всі тяжкі

window.fami



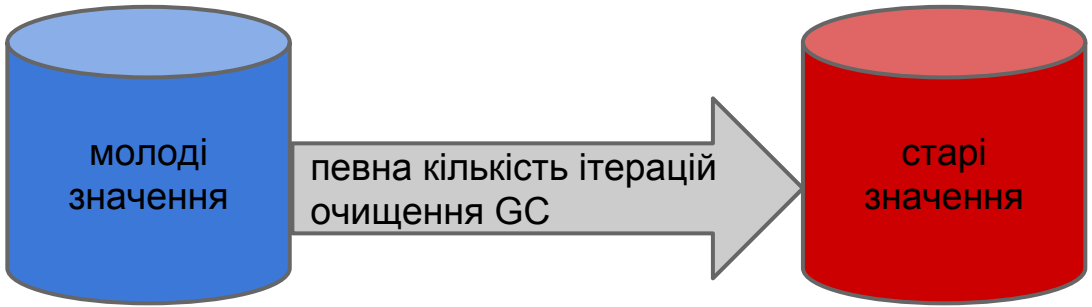
# Оптимізації

1. Старі та молоді покоління значень
2. Не доступність змінних в дев тулі

# Оптимізації

Старі та молоді покоління значень:

- значення розділяються на старі та нові
- з часом нові значення потрапдяють в секцію до старих значень



# Молоді покоління значеннь

- Швидке виділення пам’яті
- Швидка очистка
- Часта очистка

# Старі покоління значеннь

- Швидке виділення пам’яті
- Повілбніша очистка
- НЕ часта очистка

# Кроки

Спочатку дизайн
Потім код для дизайна
По ситуації чи потрібна відладка

# Як запобігти появі меморі ліків

- Не забувати видаляти  посилання на видалені DOM елементи
- Уникати циклічних посилань
- Використовувати локальний скоуп по максимуму
- Видаляти слухачі подій перед видпленням елемента
- чистити таймаути та інтервали

# Девтул

1. таск менеджер
2. тайм лайн
3. профайлер