

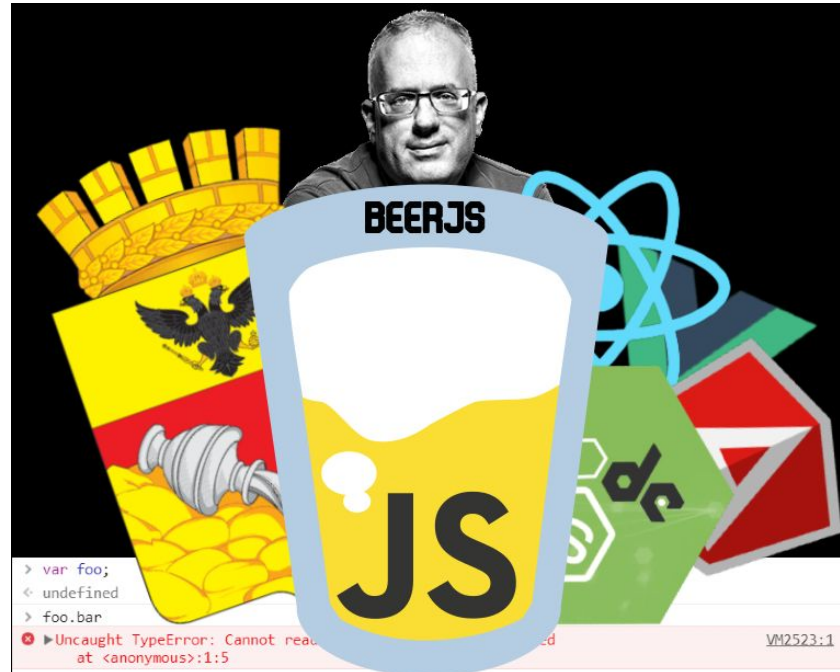
Контекстно Зависимые Топологии

(на JavaScript)



DataArt

... With many Thanks ...



... examples ...

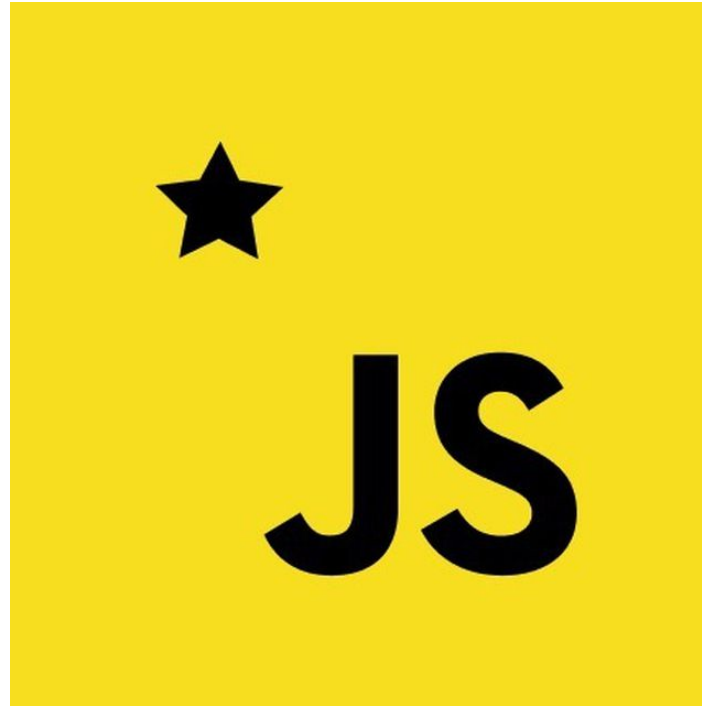


... With many Thanks ...

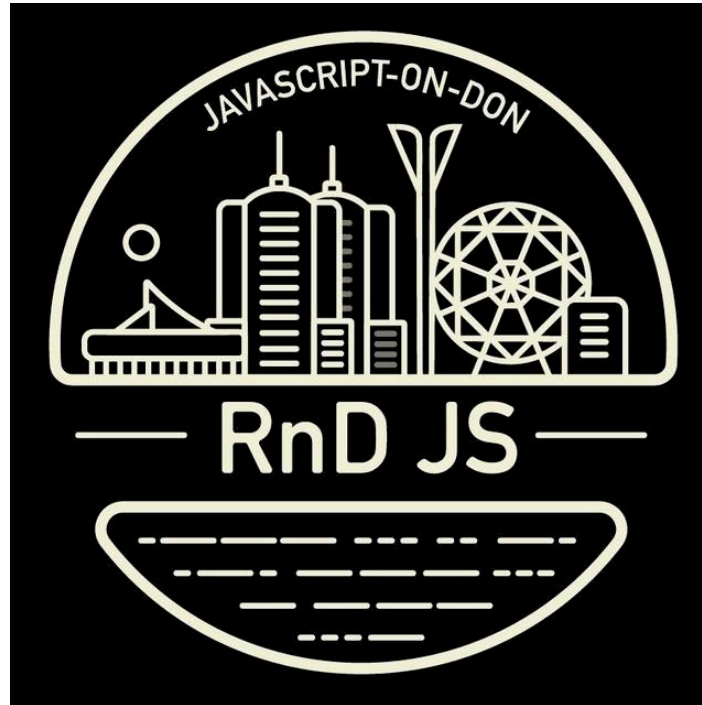


JS

... With many Thanks ...



... With many Thanks ...



... With many Thanks ...



BIO

- На JS начал писать после первого курса (1999)
- Преподавал экономику 8 лет (к.э.н.)
- Потом стал писать в «етоя» Open Source
- Надоело писать, стал рассказывать
- Рассказывать получается так же и как писать
- Но уже не могу остановиться, так как привык ...
- Памагити, послушайте ещё раз !!111





Какую проблему решаем?

null is not a mistake

my apologies to Sir Charles Antony Richard Hoare

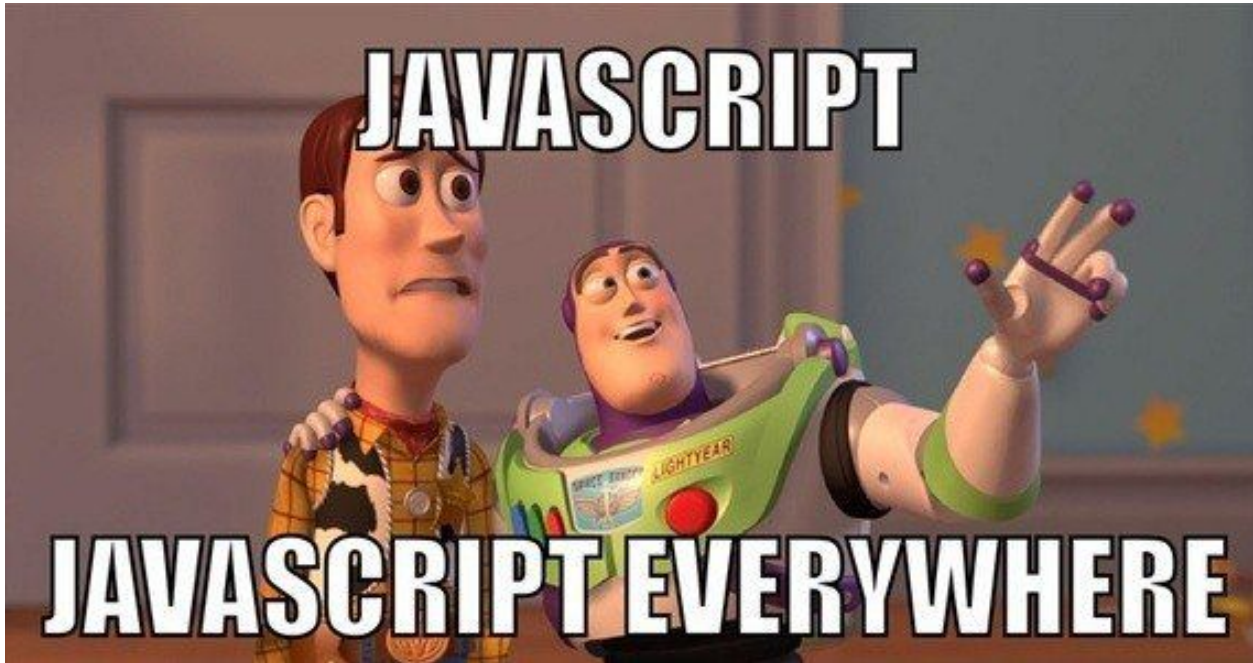


Какую проблему решаем?

type of null is also good

my apologies to Brendan Eich


Какую проблему решаем?



О чём пойдёт речь...



О чём пойдёт речь...



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Current events](#)
[Random article](#)
[About Wikipedia](#)
[Contact us](#)
[Donate](#)

[Contribute](#)
[Help](#)
[Learn to edit](#)
[Community portal](#)
[Recent changes](#)
[Upload file](#)

[Tools](#)
[What links here](#)
[Related changes](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Cite this page](#)
[Wikidata item](#)

[Print/export](#)
[Download as PDF](#)
[Printable version](#)

Topology

From Wikipedia, the free encyclopedia

*This article is about the branch of mathematics. For the mathematical structure, see [Topological space](#). For other uses, see [Topology \(disambiguation\)](#).
Not to be confused with [topography](#) or [typology](#).*

In **mathematics**, **topology** (from the [Greek](#) words τόπος, 'place, location', and λόγος, 'study') is concerned with the properties of a **geometric object** that are preserved under [continuous deformations](#), such as [stretching](#), [twisting](#), crumpling, and bending; that is, without closing holes, opening holes, tearing, gluing, or passing through itself.

A [topological space](#) is a [set](#) endowed with a structure, called a *topology*, which allows defining continuous deformation of subspaces, and, more generally, all kinds of [continuity](#). [Euclidean spaces](#), and, more generally, [metric spaces](#) are examples of a topological space, as any distance or metric defines a topology. The deformations that are considered in topology are [homeomorphisms](#) and [homotopies](#). A property that is invariant under such deformations is a [topological property](#). Basic examples of topological properties are: the [dimension](#), which allows distinguishing between a [line](#) and a [surface](#); [compactness](#), which allows distinguishing between a line and a circle; [connectedness](#), which allows distinguishing a circle from two non-intersecting circles.

The ideas underlying topology go back to [Gottfried Leibniz](#), who in the 17th century envisioned the *geometria situs* and *analysis situs*. [Leonhard Euler](#)'s [Seven Bridges of Königsberg](#) problem and [polyhedron formula](#) are arguably the field's first theorems. The term *topology* was introduced by [Johann Benedict Listing](#) in the 19th century, although it was not until the first decades of the 20th century that the idea of a topological space was developed.

Contents [hide]

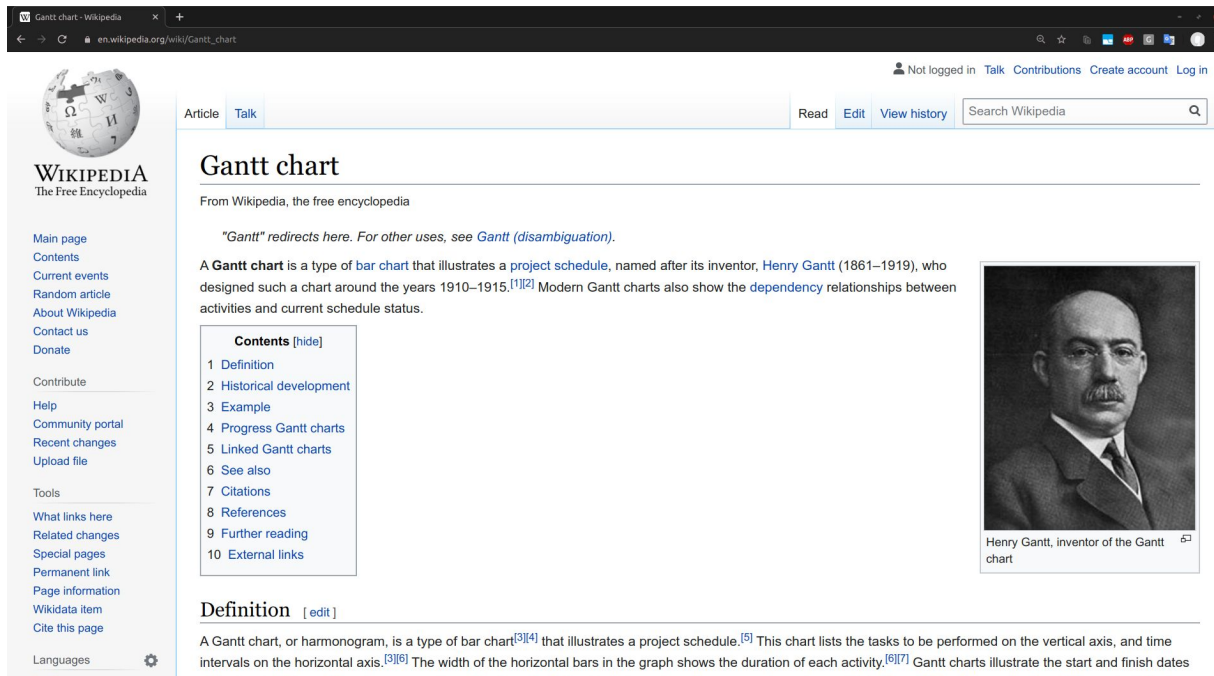
- Motivation
- History
- Concepts
 - Topologies on sets
 - Continuous functions and homeomorphisms
 - Manifolds
- Topics



[Möbius strips](#), which have only one surface and one edge, are a kind of object studied in topology.



При чём здесь Динамика Процессов ?



The screenshot shows the Wikipedia page for "Gantt chart". The page includes a sidebar with navigation links, a main content area with a table of contents, and a definition section. A portrait of Henry Gantt is also visible.

Wikipedia
The Free Encyclopedia

Not logged in | Talk | Contributions | Create account | Log in

Article | Talk | Read | Edit | View history | Search Wikipedia

Gantt chart

From Wikipedia, the free encyclopedia

"Gantt" redirects here. For other uses, see [Gantt \(disambiguation\)](#).

A **Gantt chart** is a type of [bar chart](#) that illustrates a [project schedule](#), named after its inventor, [Henry Gantt](#) (1861–1919), who designed such a chart around the years 1910–1915.^{[1][2]} Modern Gantt charts also show the [dependency](#) relationships between activities and current schedule status.

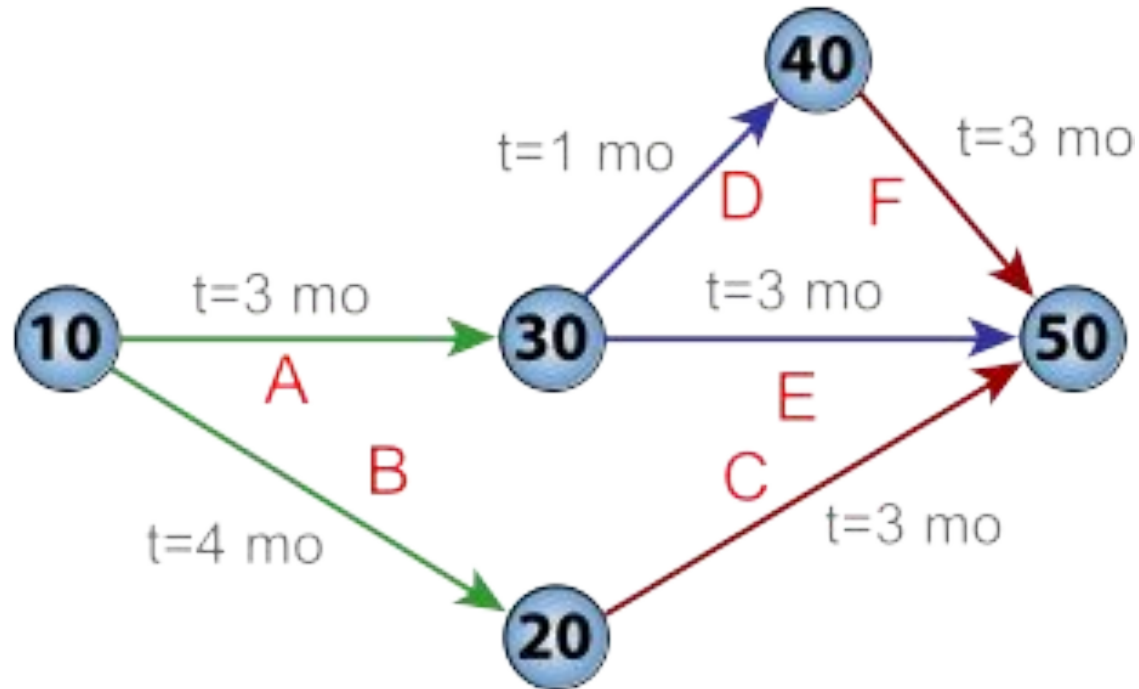
Contents [hide]
1 Definition
2 Historical development
3 Example
4 Progress Gantt charts
5 Linked Gantt charts
6 See also
7 Citations
8 References
9 Further reading
10 External links

Definition [edit]

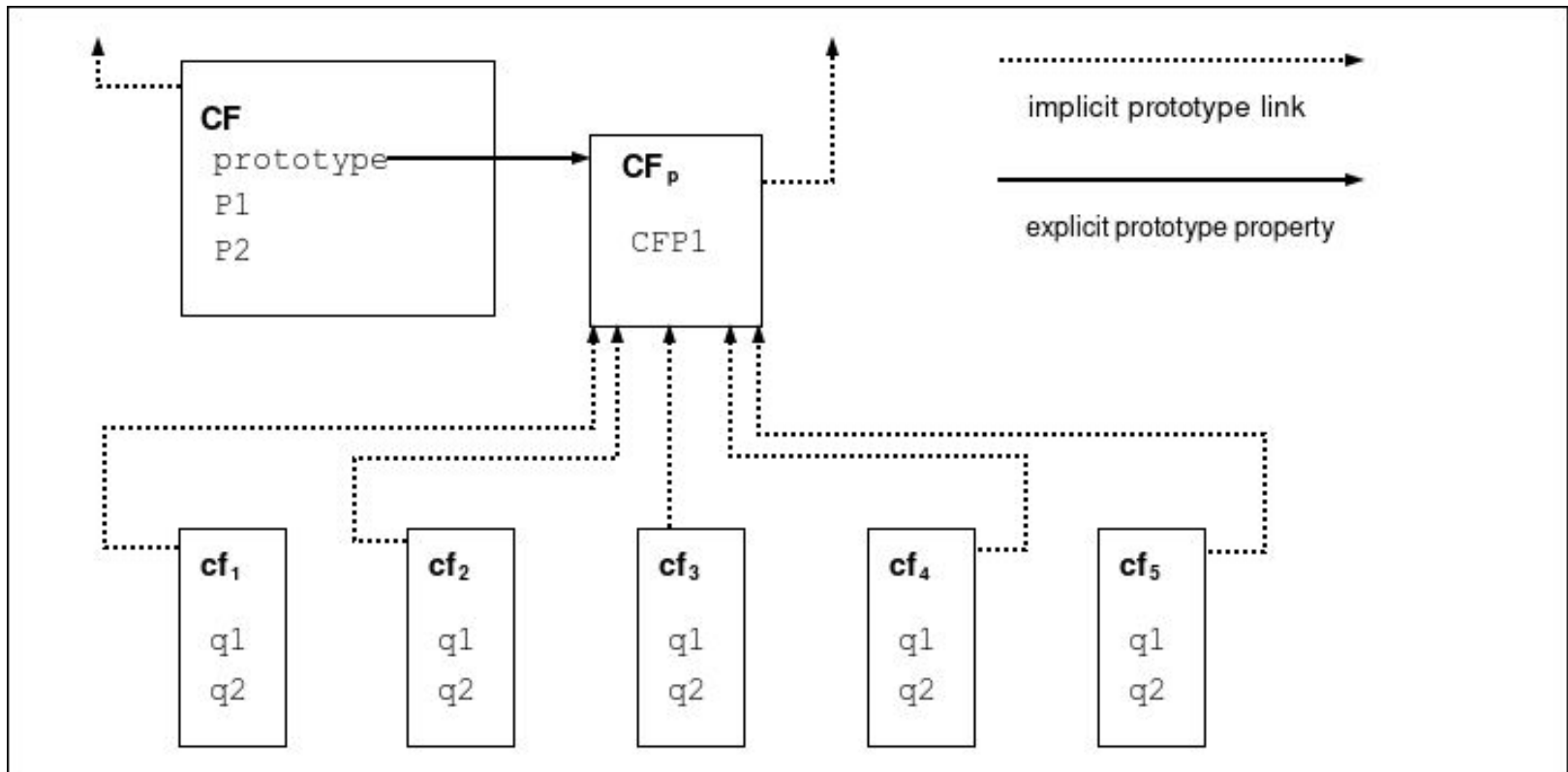
A Gantt chart, or harmonogram, is a type of bar chart^{[3][4]} that illustrates a project schedule.^[5] This chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis.^{[3][6]} The width of the horizontal bars in the graph shows the duration of each activity.^{[6][7]} Gantt charts illustrate the start and finish dates

Henry Gantt, inventor of the Gantt chart

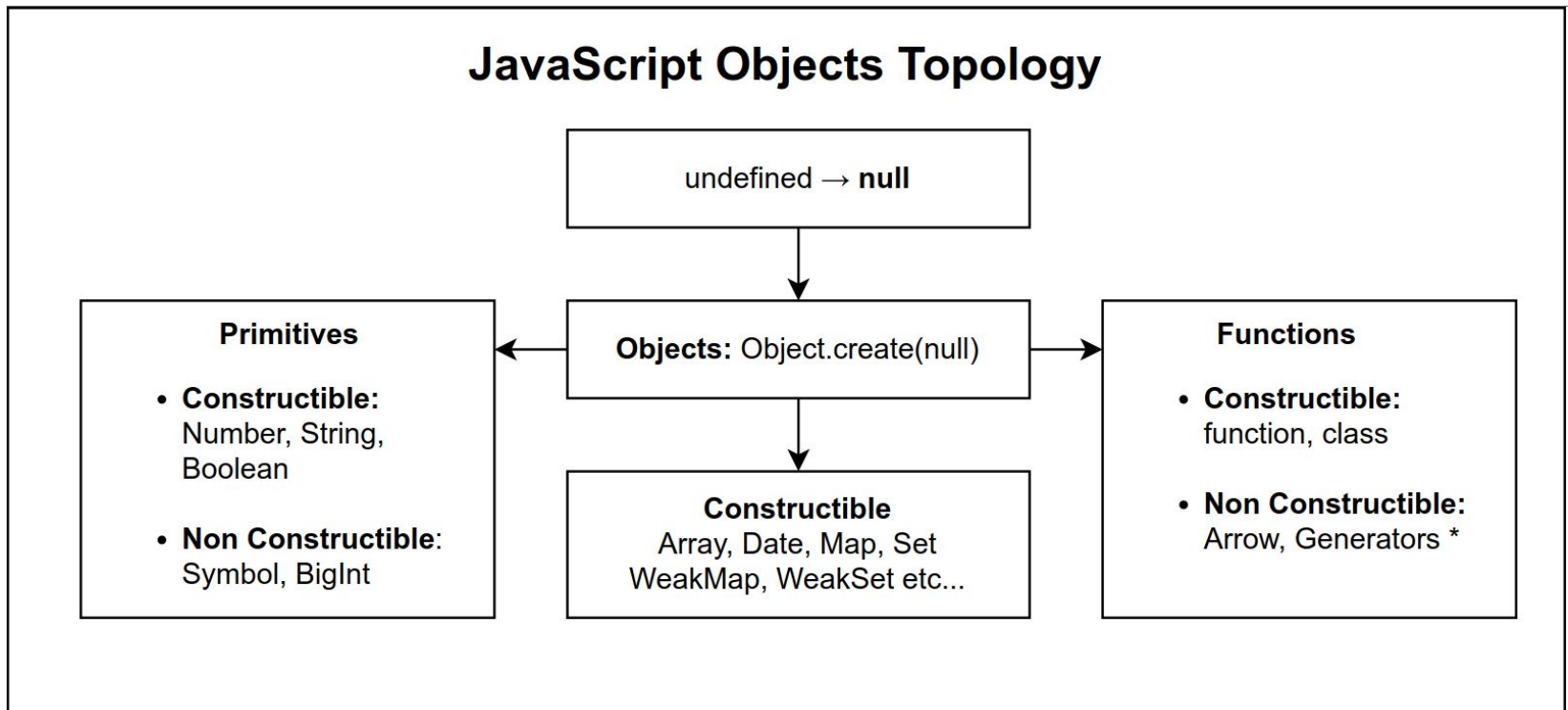
Можно ли показать Динамику в JavaScript ?



О чём, о чём : о прототипах, конечно же



О чём, о чём : о прототипах, конечно же



Как можно выразить Динамику прототипами ?

```
> next  
◀ ▼ MyConstructor {state: 3} ⓘ  
  state: 3  
  ▼ __proto__:  
    state: 2  
    ▼ __proto__:  
      state: 1  
      ▶ __proto__: Object
```

Инструментарий : Фабрика Конструкторов

```
1
2  function Factory(previous) {
3      function MyConstructor(state) {
4          this.state = state;
5      };
6      Reflect.setPrototypeOf(
7          MyConstructor.prototype, previous
8      );
9      return MyConstructor;
10 };
11
12
```

Инструментарий: Фабрика Конструкторов

```
1
2  function Factory(previous) {
3      class MyConstructor {
4          constructor(state) {
5              this.state = state;
6          }
7      };
8      Reflect.setPrototypeOf(
9          MyConstructor.prototype, previous);
10     return MyConstructor;
11 };
12
```

Что это даёт?

- Теперь Конструкторы наследуются от Экземпляров
- Runtime стал более динамичным, но сохранил суть
- Динамика происходившего лежит в Prototype Chain
- Можно добавить различные плюшки: TimeStamps
- Это наследование ... но «оно Другое»

Что это Не даёт?

- Теперь нужно будет думать над Жизненным Циклом
- С типами «всё сложно», тоже нужно будет думать
- Prototype Chain – сложная концепция, мучительно
- Можно забыть добавить различные плюшки
- Это наследование ... и «оно Другое»

Родословные Зависимых Типов

Kind (type theory)

From Wikipedia, the free encyclopedia



This article **may be too technical for most readers to understand**. Please [help improve it to make it understandable to non-experts](#), without removing the technical details. (June 2020) ([Learn how and when to remove this template message](#))

In the area of [mathematical logic](#) and [computer science](#) known as [type theory](#), a **kind** is the type of a [type constructor](#) or, less commonly, the type of a [higher-order type operator](#). A kind system is essentially a [simply typed lambda calculus](#) "one level up", endowed with a primitive type, denoted $*$ and called "type", which is the kind of any [data type](#) which does not need any [type parameters](#).

A kind is sometimes confusingly described as the "type of a [\(data\) type](#)", but it is actually more of an [arity](#) specifier. Syntactically, it is natural to consider polymorphic types to be type constructors, thus non-polymorphic types to be [nullary](#) type constructors. But all nullary constructors, thus all monomorphic types, have the same, simplest kind; namely $*$.

Since higher-order type operators are uncommon in [programming languages](#), in most programming practice, kinds are used to distinguish between data types and the types of constructors which are used to implement [parametric polymorphism](#). Kinds appear, either explicitly or implicitly, in languages whose type systems account for parametric polymorphism in a programmatically accessible way, such as [C++](#),^[1] [Haskell](#) and [Scala](#).^[2]

• Kind (Type Theory)

- **Kind** – это тип Type Constructor'a
- **Type Constructor** – свойство типизированного формального языка, которое строит **НОВЫЕ** типы из **существующих**
- Но ведь **Фабрика Конструкторов** и строит новые Конструкторы из **Экземпляров!**

• Получается, что мы Создали

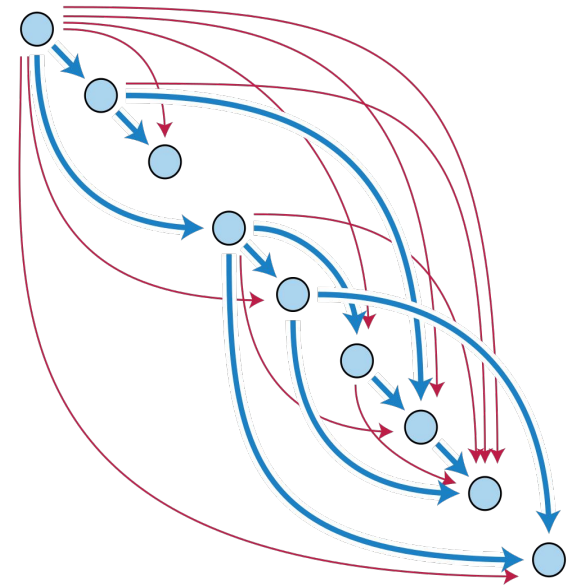
- **Конструкторы**, принимающие **полиморфный** (любой) тип в качестве параметра для своего построения
- **Типы** в Жизненном Цикле, зависящие лишь от необходимого состояния среды исполнения в настоящий момент
- Зависимости между свойствами Экземпляров типов доступные и управляемые через цепочку прототипов

• Как выглядят эти Структуры Данных

Directed acyclic graph

From Wikipedia, the free encyclopedia

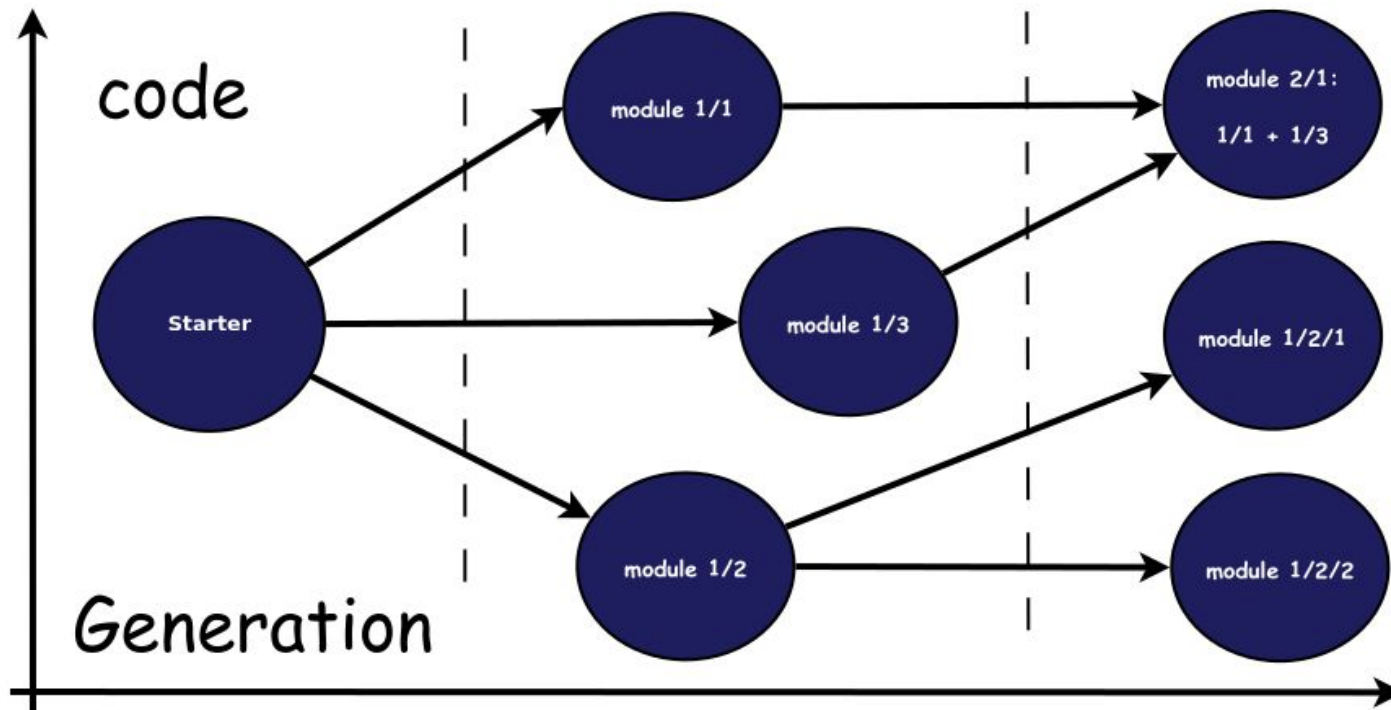
In [mathematics](#), particularly [graph theory](#), and [computer science](#), a **directed acyclic graph** (**DAG** or **dag** /ˈdæɡ/ (listen)) is a [directed graph](#) with no [directed cycles](#). That is, it consists of [vertices](#) and [edges](#) (also called *arcs*), with each edge directed from one vertex to another, such that following those directions will never form a closed loop. A directed graph is a DAG if and only if it can be [topologically ordered](#), by arranging the vertices as a linear ordering that is consistent with all edge directions. DAGs have numerous scientific and computational applications, ranging from biology (evolution, family trees, epidemiology) to sociology (citation networks) to computation (scheduling).



- Как уведомлять наследников об
Изменениях

- Не использовать одинаковые Имена Свойств:
Liskov Substitution Principle
- Определять методы Единожды в корне
цепочки: **Single Responsibility Principle**
- При необходимости подписаться на изменения:
Interface Segregation – для реализации
действий специфичных именно для данной
конкретной сущности


Инструментарий: Древовидные структуры



Инструментарий: Множественное Наследование

```
2  const a = { a: 1 };
3
4  const b = { b: 2 };
5
6  const c = { c: 3 }
7
8  const d = { d: 4 }
9
10 const e = { e: 5 }
11
12 Object.setPrototypeOf( a, b );
13 Object.setPrototypeOf( b, c );
14 Object.setPrototypeOf( c, d );
15 Object.setPrototypeOf( d, e );
```

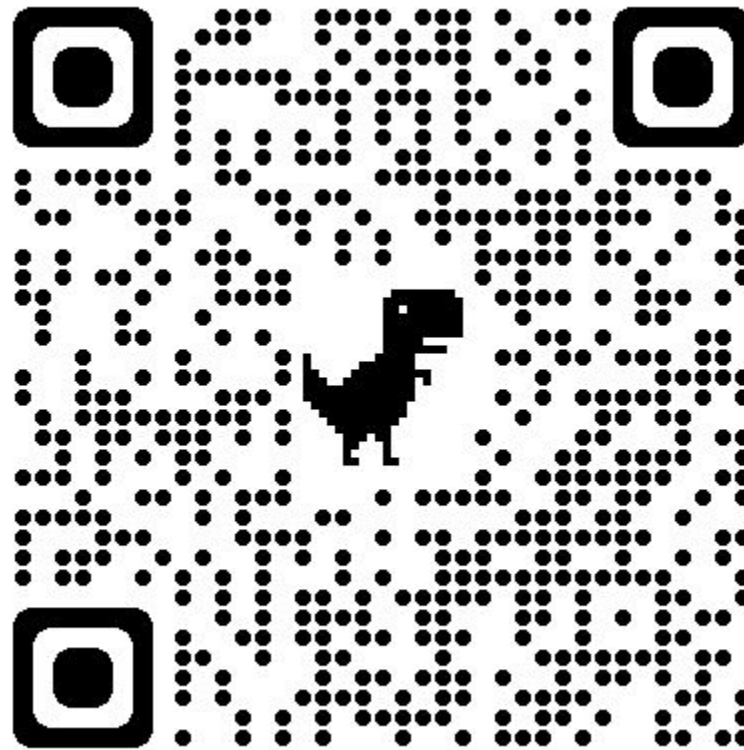
Инструментарий: Кольцевая структура Наследования

```
106  const ring = (...args) => {  
107        
108      const initial = proxify(args[0]);  
109      let current = initial;  
110  
111      args.slice(1).forEach(arg => {  
112          const p = proxify(arg);  
113          Object.setPrototypeOf(current, p);  
114          current = p;  
115      }, current);  
116  
117      Object.setPrototypeOf(current, initial);  
118  
119      return initial;  
120  }  
121  };
```



Спасибо!

... examples ...



FIN