أساسيات البرمجة

مبادى البرمجة

لماذا نتعلم البرمجة ؟

- تطور طريقة التفكير الممنهج، والإبداعي.
- البرمجة جزءًا لا يتجزأ من كل صناعة.
- العمل عن بعد.



ما الذي احتاجه ؟



الفضول

الرغبة

الخيال

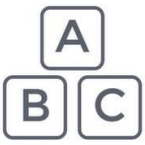
لغة الآلة Machine Code

- عبارة عن واحدات وأصفار.
- يفهمها الحاسوب.
- صعبة القراءة والتعديل.

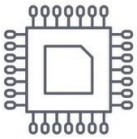
0101110
0111010
0101110

تنقسم لغات البرمجة الى قسمين تبعا لقربها للغة الالة

تصنيف لغات البرمجة



عالية المستوى



منخفضة المستوى

اللغات عالية المستوى

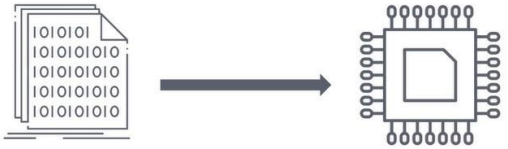
منصات تشغيل البرامج

ملاحظة : لغة swift لتطبيقات iosو kotlin لأندرويد أيضا لغة جافا ممكن أن تسخدم في تطوير تطبيقات الجوال

مستويات لغات البرمجة

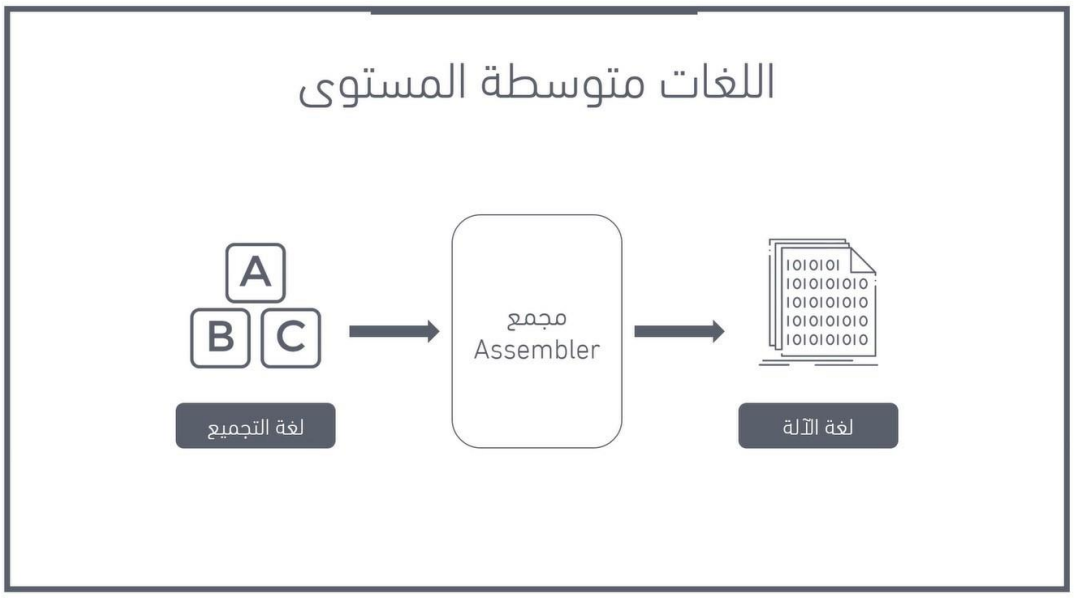
- لغات منخفضة المستوى (Low Level).
- لغات متوسطة المستوى (Medium Level).
- لغات عالية المستوى (High Level).
- لغات راقية (High-end Level).

اللغات منخفضة المستوى

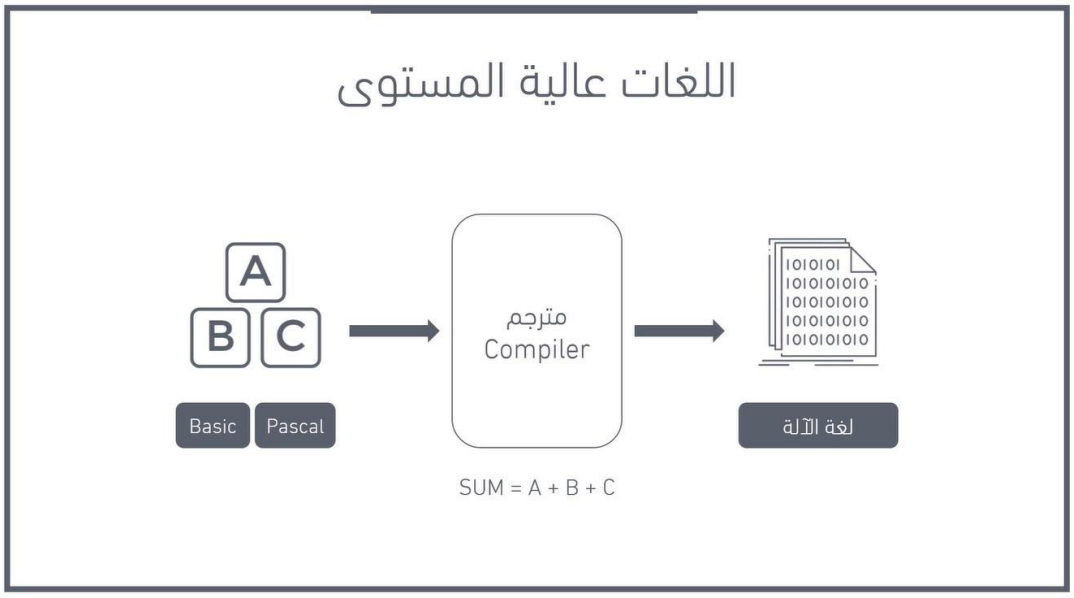


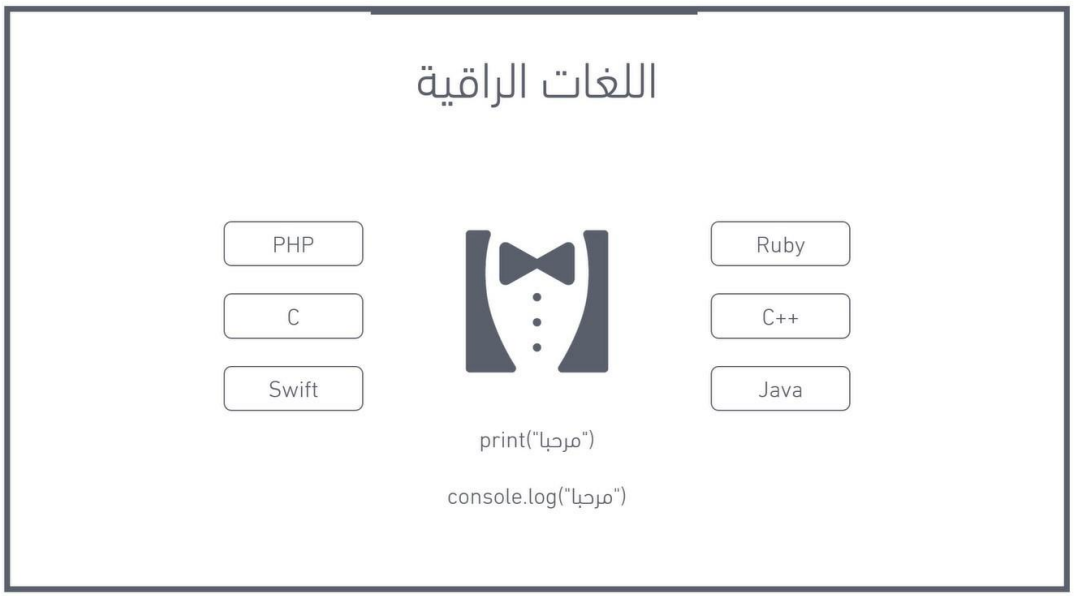
لغة الآلة

المعالج



لغة التجميع (Assembly) تحتاج إلى **"مُجَمِّع"**، فالمقصود بـ **المُجَمِّع (Assembler)** هو **برنامج** يقوم بترجمة أو تحويل التعليمات المكتوبة بلغة التجميع إلى **لغة الآلة (Machine Code)**التي يفهمها المعالج مباشرة.





ملاحظة: يقصد بالمحمولية هو:قابلة للتنقل بين الأجهزة وأنظمة التشغيل المختلفة

المقارنة	لغات منخفضة المستوى (Low-level)	لغات عالية المستوى (High-level)
التعامل مع العتاد (الهاردوير)	قريبة جدًا من العتاد مثل المعالج والذاكرة والسجلات	بعيدة عن العتاد، تتعامل مع الأفكار والمنطق البرمجي
سهولة الفهم والكتابة	صعبة ومعقدة، تحتاج معرفة تفصيلية ببنية المعالج	سهلة القراءة والكتابة، قريبة من اللغة البشرية
التحكم في الموارد	توفر تحكمًا دقيقًا جدًا في الذاكرة والعمليات	لا توفر تحكمًا مباشرًا، النظام يدير الموارد تلقائيًا
السرعة في التنفيذ	أسرع لأنها تُترجم مباشرة إلى لغة الآلة	أبطأ نسبيًا بسبب وجود طبقات ترجمة إضافية
الترجمة أو التنفيذ	تحتاج إلى مُجَمّع (Assembler)	تحتاج إلى مُترجِم (Compiler) أو مُفسِّر (Interpreter)
قابلية النقل بين الأجهزة	غير قابلة للنقل (مرتبطة بمعالج أو نظام محدد)	قابلة للنقل بسهولة بين أنظمة مختلفة
أمثلة	لغة الآلة، لغة التجميع (Assembly)	Python ،C ،Java ،C++ ، JavaScript

محررات شيفرة (Code Editors) مشهورة، وهذه نبذة مختصرة عن كل واحد منها:

---

1. Visual Studio Code (VS Code)

- من تطوير **Microsoft**.
- محرر مجاني ومفتوح المصدر.

- يدعم العديد من لغات البرمجة عبر الإضافات.(Extensions)
- يحتوي على ميزات قوية مثل التكملة التلقائية، التصحيح(Debugging) ، وتكامل.Git

---

🟧 2. Sublime Text

- محرر خفيف وسريع جدًا.
- مشهور بواجهة بسيطة وسرعة عالية في التعامل مع الملفات الكبيرة.
- يدعم الإضافات لكنه ليس مفتوح المصدر (نسخة مدفوعة مع فترة تجريبية).

🟢 3. Atom

- من تطوير (**GitHub**الذي تملكه Microsoft الآن.(
- مفتوح المصدر وقابل للتخصيص الكامل.
- يعتمد على تقنيات الويبHTML) ، CSS، .(JavaScript
- توقف تطويره رسميًا في ٢٠٢٢، لكن لا يزال يُستخدم من قبل البعض.

🟦 4. WebStorm

- من تطوير شركة .**JetBrains**
- محرر احترافي مخصص لتطوير **تطبيقات الويب** باستخدام JavaScript و TypeScriptو Reactوغيرها.
- يحتوي على أدوات مدمجة قوية لتحليل الشيفرة وتصحيحها.
- **مدفوع**، لكن يقدم فترة تجريبية مجانية.

💡ما هي لغةJavaScript ؟

JavaScriptهي **لغة برمجة عالية المستوى** تُستخدم في الأساس لجعل **صفحات الويب تفاعلية**.
تم تطويرها في الأصل لتعمل **داخل المتصفح**، لكنها أصبحت اليوم تُستخدم في **مجالات كثيرة خارج المتصفح** أيضًا.

🧭فائدةJavaScript

- تضيف **ديناميكية وتفاعلًا** للمواقع الإلكترونية.
- تسمح بتغيير محتوى الصفحة بدون إعادة تحميلها.
- تُستخدم في إنشاء **واجهات المستخدم (Front-End)** الحديثة.

🔷مثال بسيط:
عند الضغط على زر فتظهر نافذة أو يتغير لون العنصر — هذا غالبًا يتم بواسطة **JavaScript**.

🌐أين تعملJavaScript ؟

في الأصل، كانت تعمل فقط **داخل المتصفح (Browser)** مثل:

- Google Chrome
- Firefox
- Safari

حيث يقوم **محرك JavaScript** المدمج في المتصفح) مثل V8 في (Chrome بتنفيذ الشيفرة.

⚙️كيف أصبحت تعمل خارج المتصفح؟

مع ظهور بيئة التشغيل **Node.js**عام ٢٠٠٩، أصبح بالإمكان تشغيل JavaScript **خارج المتصفح**.

Node.js 🔷يستخدم نفس محرك **V8**الموجود فيChrome ، لكنه أضاف أدوات تسمح للغة بالتعامل مع:

- الملفات في نظام التشغيل
- الشبكات والإنترنت
- قواعد البيانات

وبذلك أصبحت JavaScript **لغة شاملة** يمكنها العمل على **الخوادم** وليس فقط في المتصفح.

💻ماذا يعني "تعمل في طرف الخادم"؟

لفهم ذلك، نميز بين طرفين في أي تطبيق ويب:

المصطلح	المعنى
طرف العميل (Client-side)	ما يُنفذ في جهاز المستخدم، مثل واجهة الموقع، الأزرار، النصوص، والتفاعل معها.
طرف الخادم (Server-side)	ما يُنفذ في الخادم الذي يستضيف الموقع أو التطبيق — مثل استقبال الطلبات، معالجة البيانات، التواصل مع قاعدة البيانات، ثم إرسال النتائج للمستخدم.

🔷 عندما نقول إن **JavaScript تعمل في طرف الخادم**، فهذا يعني أنها:

تعالج الطلبات وتتعامل مع قواعد البيانات على الخادم نفسه باستخدام) Node.js بدون الحاجة لمتصفح.(

🛠️استخدامات JavaScript اليوم

١. **تطوير واجهات المواقع – (Front-End)** باستخدام مكتبات مثل:

- React.js
- Vue.js
- Angular

٢. **تطوير الخوادم – (Back-End)** باستخدام:

- Node.js
- Express.js

٣. **تطبيقات الجوال – (Mobile Apps)** باستخدام:

- React Native
- Ionic

٤. **تطبيقات سطح المكتب – (Desktop Apps)** باستخدام:

- Electron.js (مثلاً تطبيق VS Code مبني عليه(

٥. **الذكاء الاصطناعي والروبوتات**)بشكل محدود حاليًا.(

🚀**أشهر التطبيقات والمواقع المبنية بـJavaScript**

- **Facebook / Instagram** → واجهاتReact.js
- **Netflix** → يستخدم Node.js في الخوادم
- **LinkedIn** → يستخدم Node.js في الخدمات الخلفية
- **Uber** → يعتمد على Node.js في نظامه الفوري
- **Visual Studio Code** → مبني على Electron (JavaScript + HTML + CSS)

🧩**خلاصة سريعة**

العنصر	التوضيح
اللغة	JavaScript
الأصل	كانت للمتصفح فقط
الآن	تعمل في المتصفح والخادم
البيئة التي جعلتها تعمل خارج المتصفح	Node.js
المجالات	الويب، الخوادم، الجوال، سطح المكتب
أهم المكتبات	React، Vue، Node.js، Express
أشهر تطبيقات مبنية بها	Netflix، Facebook، Uber، VS Code

الفرق بين عملها **داخل المتصفح** و**خارج المتصفح**)في(Node.js ، مع أمثلة توضيحية 👇

🟩**أولاً :طريقة عمل JavaScript داخل المتصفح(Front-End)**

الفكرة:

عندما تفتح صفحة ويب، المتصفح يقوم بتحميل ملفات:

- **HTML** ← هيكل الصفحة
- **CSS** ← الشكل والتنسيق
- **JavaScript** ← التفاعل والسلوك

ثم يقوم **محرك JavaScript داخل المتصفح** (مثل **V8** في Chrome أو **SpiderMonkey** في Firefox) بتفسير وتنفيذ أوامر JavaScript خطوة بخطوة.

كيف تعمل فعليًا:

أنشئ ملف نصي بامتداد html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script src="script.js"></script>
</body>
</html>
```

ثم أنشئ ملف اخر بامتداد js واسمه مثلا script واكتب فيه:-

```
alert("hello world")
console.log("hello world")
```

ما يحدث:

- المتصفح يقرأ كود JavaScript.
- لا يستطيع الوصول إلى الملفات أو النظام، حفاظًا على أمان المستخدم.

✅**النتيجة:**

الكود يُنفذ في جهاز المستخدم، ويتعامل فقط مع الصفحة نفسها.

هذا هو **العمل داخل المتصفح = الواجهة الأمامية.(Front-End)**

🟦**ثانيًا :طريقة عمل JavaScript خارج المتصفح(Back-End)**

💡**الفكرة:**

Node.js"هو بيئة تشغيل JavaScript على السيرفر، تسمح لك بتشغيل أكواد JavaScriptخارج المتصفح لإنشاء تطبيقات ويب وخوادم وبرامج متنوعة".

عند تثبيت **Node.js**، يصبح بإمكاننا تشغيل ملفات js.على جهازنا أو على الخادم بدون الحاجة إلى متصفح.

Node.jsيستخدم نفس محرك **V8**الموجود فيChrome ، لكنه يضيف له مكونات إضافية تسمح بـ:

- التعامل مع الملفات
- تشغيل الخوادم
- التواصل مع الإنترنت
- التعامل مع قواعد البيانات

⚙️**مثال عملي في:Node.js**

ادخل على سطر الأوامر cmdثم اكتب nodeطبعا بعد تحميلها من الموقع الرسمي وتثبيتها في النظام ثم اكتب مثلا هذا الكود:-

console.log("hello world")

📘**ما يحدث:**

- الكود يُنفذ في بيئة) Node.js وليس في متصفح.(
- ينشئ **خادم ويب** يستقبل الطلبات ويرد عليها بالنص المحدد.
- الآن JavaScript أصبحت تتعامل مع **الخادم، الملفات، الإنترنت** مباشرة.

✅**النتيجة:**
الكود يُنفذ في الخلفية (الخادم)، ويخدم المستخدمين عبر الإنترنت.

هذا هو **العمل خارج المتصفح = الواجهة الخلفية.(Back-End)**

✅**مقارنة سريعة بين البيئتين:**

العنصر	**داخل المتصفح**	**خارج المتصفح (Node.js)**
مكان التنفيذ	جهاز المستخدم	الخادم أو جهاز المطور
التعامل معHTML/CSS	✅ممكن	❌ غير ممكن
الوصول إلى الملفات والنظام	❌ غير ممكن	✅ممكن
المكتبات المتاحة	DOM, BOM, Fetch API	File System, HTTP, OS
الاستخدام الشائع	واجهات المستخدم(Front-End)	خوادم الويب(Back-End)

🎯**الخلاصة:**

- JavaScriptفي الأصل صُممت لتعمل **في المتصفح** للتفاعل مع صفحات الويب.
- ثم ظهرت **Node.js**لتجعلها تعمل **خارج المتصفح** في الخوادم والتطبيقات الخلفية.
- اليوم، JavaScriptأصبحت **لغة موحدة** يمكنها بناء كل شيء:
  🌐الواجهة الأمامية + ⚙️ الواجهة الخلفية + 📱 تطبيقات الجوال + 💻 تطبيقات سطح المكتب.

المتغيرات

Variables

🧠أولًا: ما هو المتغير؟

المتغير هو مساحة في الذاكرة نستخدمها لتخزين قيمة يمكن تغييرها أثناء تنفيذ البرنامج.
في JavaScript نستخدم الكلمات المفتاحية let, const, وvar لتصريح المتغيرات.

🏷️قواعد تسمية المتغيرات والثوابت

١. **يجب أن تبدأ باسم يتكوّن من:**
   - حرف (A–Z أو a–z)
   - أو علامة الدولار $
   - أو الشرطة السفلية _
٢. **لا يمكن أن تبدأ برقم.**
٣. **الأسماء حساسة لحالة الأحرف** (case-sensitive)
   name 👉و Nameمتغيران مختلفان.
٤. **لا يمكن استخدام الكلمات المحجوزة** مثل...if, for, class, return :
٥. **يفضّل اتباع نمط الكتابة camelCase** مثل.userName, totalPrice : على النمط الاخر وهو snake_case بحيث تفصل الكلمات بشرطة سفلية مثل user_name
٦. **الثوابت (const)** تُكتب عادةً بأحرف كبيرة مع الشرطات السفلية:
   const MAX_SPEED = 120;

الكلمة	إعادة التعريف	تغيير القيمة	ملاحظات
var	ممكن	ممكن	قديمة – يُفضل عدم استخدامها
let	غير ممكن	ممكن	الأفضل للاستخدام في المتغيرات المتغيرة
const	غير ممكن	غير ممكن (إلا تعديل خصائص الكائن)	الأفضل للثوابت

🧩 **ملاحظة:**
حتى مع const، يمكن تعديل **محتوى الكائن أو المصفوفة**، لكن لا يمكن إعادة إسناد المتغير نفسه:

```
const user = {name: "Ali"};
user.name = "Omar"; // ✅ مسموح
user = {}; // ❌ خطأ
```

🔢 الأنواع البدائية(Primitive Types)

هي الأنواع الأساسية فيJavaScript

النوع	مثال	الوصف
String	"Hello"	تمثل النصوص
Number	42 , 3.14	الأرقام (صحيحة أو عشرية)
Boolean	true , false	القيم المنطقية
Undefined	—	متغير تم تعريفه بدون قيمة
Null	null	لا شيء (فراغ مقصود)
BigInt	123n	أعداد صحيحة كبيرة جدًا
Symbol	Symbol("id")	معرف فريد وغير قابل للتكرار

🧩الأنواع المتقدمة — (Reference Types)

هذه تُخزّن **بالمراجع**:**(by reference)**

- **Object** → كائن يحتوي على أزواج (مفتاح:قيمة)
- **Array** → مصفوفة من القيم [1,2,3]
- **Function** → دالة يمكن استدعاؤها
- **Date, Map, Set, WeakMap, WeakSet, Class** ... إلخ.

⚙️الفرق بين اللغات الساكنة (Static) والديناميكية(Dynamic)

النوع	التوصيف	مثال
اللغات الساكنة (Static)	يجب تحديد نوع المتغير عند التصريح به، ولا يمكن تغييره لاحقًا	Java, C++
اللغات الديناميكية (Dynamic)	لا يُلزمك بتحديد نوع المتغير، والنوع يمكن أن يتغير أثناء التشغيل	JavaScript, Python

🧩مثال في:JavaScript

```
let x = 5;        // رقم
x = "Hello";      // يتحول إلى نص – لا مشكلة
```

بينما في) Java الساكنة:(

```
int x = 5;
x = "Hello"; // ❌ خطأ في وقت الترجمة
```

💡إضافات مهمة

- يمكنك معرفة نوع المتغير باستخدام :typeof

```
typeof "Hello"; // "string"
typeof 123;     // "number"
```

العوامل

Operators

العوامل (Operators) هي رموز تُستخدم لتنفيذ عمليات على القيم (المتغيرات أو الثوابت).

---

1. 🔷العوامل الرياضية(Arithmetic Operators)

تُستخدم لإجراء العمليات الحسابية.

العامل	الوصف	المثال	النتيجة
+	الجمع	5 + 2	7
-	الطرح	5 - 2	3
*	الضرب	5 * 2	10
/	القسمة	10 / 2	5
%	باقي القسمة	7 % 3	1
**	الأس (الرفع للقوة)	2 ** 3	8
++	زيادة بمقدار ١	x++	تزيد القيمة ١ بعد التنفيذ
--	نقصان بمقدار ١	x--	تنقص القيمة ١ بعد التنفيذ

ملاحظة:

++xيختلف عن :x++

- x++: يطبع ثم يزيد القيمة
- ++x: يزيد القيمة ثم يطبع

---

2. ◆ عوامل الربط(Concatenation Operator)

تُستخدم لربط النصوص (Strings) باستخدام .+

المثال	النتيجة
"Hello " + "World"	"Hello World"
"Age: " + 20	"Age: 20"

إذا استخدمت +بين نص ورقم، **سيتم تحويل الرقم إلى نص تلقائيًا**.

---

3. ◆ عوامل الإسناد(Assignment Operators)

تُستخدم لإسناد القيم إلى المتغيرات، ويمكن دمجها مع العمليات الحسابية.

العامل	الوصف	المثال	ما يعادلها
=	إسناد قيمة	x = 10	—
+=	جمع وإسناد	x += 5	x = x + 5
-=	طرح وإسناد	x -= 5	x = x - 5
*=	ضرب وإسناد	x *= 2	x = x * 2
/=	قسمة وإسناد	x /= 2	x = x / 2
%=	باقي قسمة وإسناد	x %= 3	x = x % 3
**=	أس وإسناد	x **= 2	x = x ** 2

---

4. 🔷 عوامل المقارنة(Comparison Operators)

تُستخدم لمقارنة القيم وتُرجع نتيجة **منطقية (true** أو.**false)**

العامل	الوصف	المثال	النتيجة
>	أكبر من	5 > 3	true
<	أصغر من	2 < 3	true
>=	أكبر من أو يساوي	5 >= 5	true
<=	أصغر من أو يساوي	4 <= 3	false

---

5. 🔷 عوامل المساواة(Equality Operators)

العامل	الوصف	المثال	النتيجة
==	يساوي (مع تحويل النوع تلقائيًا)	5 == "5"	true
===	يساوي تمامًا (نفس القيمة ونفس النوع)	5 === "5"	false
!=	لا يساوي (مع تحويل النوع)	5 != "5"	false
!==	لا يساوي تمامًا (بدون تحويل النوع)	5 !== "5"	true

⚠ **نصيحة:** استخدم دائمًا === و !== لتجنّب الأخطاء الناتجة عن التحويل التلقائي للأنواع.

---

6. 🔷 العوامل المنطقية (Logical Operators)

تُستخدم للعمليات المنطقية على القيم **true / false**.

العامل	الوصف	المثال	النتيجة
&&	AND (و) – تُرجع true إذا كان الشرطان صحيحين	`console.log((5 > 2) && (3 < 4))`	true
||	OR (أو) – تُرجع true إذا كان أحد الشرطين صحيحًا	`console.log((2 > 5) || (3 < 4))`	true
!	NOT (نفي) – تعكس القيمة	`console.log(!(5 > 2))`	false

---

7. 🔷 العوامل الثنائية (Bitwise Operators)

تتعامل مع **البتات (bits)** في الأرقام الثنائية (binary).

العامل	الوصف	المثال	النتيجة (بالثنائي)
&	AND	5 & 1	0101 & 0001 = 0001 → 1
||	or	5 & 1	0101 || 0001 = 0101 → 5

💡 ملاحظات إضافية:

- يمكنك استخدام الأقواس ( ) لتحديد أولوية التنفيذ.
- أولوية العمليات:
  1. الأقواس
  2. العمليات الرياضية
  3. المقارنة
  4. المنطقية
  5. الإسناد

أولًا: الكائنات (Objects) في JavaScript

🔷 ما هو الكائن؟

الكائن هو مجموعة من الخصائص (properties)، وكل خاصية تتكون من اسم (key) وقيمة (value).

```
let person = {
  name: "Ali",
  age: 25,
  greet: function() {
    console.log("Hello, my name is " + this.name);
  }
};

console.log(person.name); // "Ali"
person.greet(); // "Hello, my name is Ali"
```

🔶 الكائن يمكن أن يحتوي على بيانات (متغيرات) ووظائف (دوال)

🔶 الوظائف داخل الكائن تُسمى طرق (methods)

🧩 ثانيًا: مبادئ البرمجة الكائنية (OOP) في JavaScript

الـ OOP تقوم على أربع ركائز أساسية:

التغليف (Encapsulation)

التجريد (Abstraction)

الوراثة (Inheritance)

تعدد الأشكال (Polymorphism)

دعنا نشرح كل مبدأ بلغة بسيطة وأمثلة 👇

1️⃣ التغليف (Encapsulation)

يعني تجميع البيانات والوظائف المرتبطة بها في كائن واحد، مع إخفاء التفاصيل الداخلية عن الخارج.

ببساطة: كل كائن يحافظ على بياناته، ولا يُسمح بالوصول إليها إلا عبر واجهات محددة (methods).

```
class BankAccount {
  #balance = 0; // خاص (private)

  deposit(amount) {
    this.#balance += amount;
  }

  getBalance() {
    return this.#balance;
  }
}

let account = new BankAccount();
account.deposit(100);
console.log(account.getBalance()); // ✅ 100
// console.log(account.#balance); ❌ خطأ: خاص لا يمكن الوصول إليه
```

💡 الفائدة: منع العبث ببيانات الكائن من الخارج.

2 التجريد (Abstraction)

يعني إخفاء التفاصيل المعقدة وإظهار فقط الوظائف المهمة للمستخدم.

أي أنك تُخفي طريقة عمل الأشياء داخليًا، وتُظهر فقط كيف تُستخدم.

```
class Car {
  startEngine() {
    console.log("Engine started");
  }
  drive() {
    console.log("Driving...");
  }
}

let car = new Car();
car.startEngine(); // لا تحتاج تعرف كيف تعمل الآلة، فقط تستخدمها
car.drive();
```

💡 الفائدة: تسهّل التعامل مع الكائنات دون الحاجة لفهم التفاصيل الداخلية.

3 الوراثة (Inheritance)

تعني أن كائنًا جديدًا (أو صنفًا) يمكنه وراثة خصائص ودوال من كائن آخر.

```
class Animal {
  eat() {
    console.log("Eating...");
  }
}

class Dog extends Animal {
```

```
  bark() {
    console.log("Woof!");
  }
}

let dog = new Dog();
dog.eat();  // ✅ موروثة من Animal
dog.bark(); // ✅ خاصة بـ Dog
```

💡 الفائدة: إعادة استخدام الكود وتجنّب التكرار.

4️⃣ تعدد الأشكال (Polymorphism)

يعني أن الدوال يمكن أن تتصرف بشكل مختلف حسب نوع الكائن الذي يستدعيها.

ببساطة: نفس الاسم، لكن تنفيذ مختلف.

```
class Animal {
  makeSound() {
    console.log("Some sound");
  }
}

class Cat extends Animal {
  makeSound() {
    console.log("Meow");
  }
}

class Dog extends Animal {
  makeSound() {
    console.log("Woof");
  }
}
```

```
let animals = [new Cat(), new Dog(), new Animal()];
animals.forEach(a => a.makeSound());
/*
Meow
Woof
Some sound
*/
```

💡 الفائدة: مرونة في التعامل مع الكائنات المختلفة بنفس الواجهة.

⚙️ مثال يجمع كل المفاهيم السابقة

```
class User {
  #password; // تغليف (خاص)

  constructor(name, password) {
    this.name = name;
    this.#password = password;
  }

  login(inputPassword) { // تجريد (واجهة بسيطة)
    return inputPassword === this.#password;
  }
}

class Admin extends User { // وراثة
  deleteUser(user) {
    console.log(`${user.name} deleted by admin.`);
  }
}

class Guest extends User {
  login() { // تعدد الأشكال (سلوك مختلف)
    console.log("Guests cannot log in.");
  }
}

let admin = new Admin("Ali", "1234");
let guest = new Guest("Visitor");
```

```
console.log(admin.login("1234")); // true
guest.login(); // "Guests cannot log in."
admin.deleteUser(guest); // "Visitor deleted by admin."
```

🎯 الخلاصة

الفائدة	التعريف المختصر	المبدأ
الأمان والتنظيم	حماية البيانات داخل الكائن	**التغليف**
البساطة وسهولة الاستخدام	إخفاء التفاصيل الداخلية	**التجريد**
إعادة استخدام الكود	كائن يرث خصائص من كائن آخر	**الوراثة**
المرونة	نفس الدالة تتصرف بشكل مختلف	**تعدد الأشكال**

أولاً: أنواع الأخطاء فيJavaScript

1.الأخطاء القواعدية(Syntax Errors)

- تحدث عندما يكتب المبرمج الكود بطريقة تخالف قواعد اللغة.
- تمنع المترجم من تنفيذ الكود إطلاقًا.

أمثلة:

```
let x = 10
if (x > 5) {  // نسي إغلاق القوس
  console.log("x كبير");
```

النتيجة:

SyntaxError: Unexpected end of input

طرق المعالجة:

- استخدام **محرر ذكي** مثل VS Code يوضح الأخطاء القواعدية مباشرة.

- مراجعة الأقواس والفواصل المنقوطة والمسافات.
- تشغيل الكود في **Console** لمتصفحك لمعرفة موضع الخطأ.

2.أخطاء زمن التشغيل(Runtime Errors)

- تحدث أثناء تنفيذ الكود بعد أن يمر من مرحلة الترجمة بنجاح.
- تحدث عادة بسبب مشاكل في القيم أو المتغيرات أو استدعاء دوال غير معرفة.

أمثلة:

```
let num = 5;
console.logo(num); // الدالة كتبت بشكل خاطئ
```

النتيجة:

Uncaught TypeError: console.logo is not a function

طرق المعالجة:

- استخدام **try...catch** لمعالجة الأخطاء بدون توقف البرنامج:

```
let num = 5;
try{
    console.loge(num) // كتبت الدالة بشكل خاطىء
}catch(error){
    console.log("There is an", error)
}
```

- التحقق من نوع المتغير قبل استخدامه.(typeof num)
- طباعة القيم في نقاط مختلفة بـ console.log()لتتبع مكان الخطأ.

3.الأخطاء المنطقية(Logical Errors)

- البرنامج يعمل دون توقف، لكن **النتائج تكون خاطئة** بسبب منطق غير صحيح.
- لا تظهر رسالة خطأ في الـConsole.

أمثلة:

```
let price = 100;

let discount = 0.2;
let total = price + discount; // المفروض price - discount
```

```
console.log(total); // الناتج ١٠٠,٢ بدلاً من ٨٠
```

طرق المعالجة:

- مراجعة منطق البرنامج خطوة بخطوة.
- استخدام **console.log()** لعرض القيم المتغيرة أثناء التنفيذ.
- كتابة **اختبارات (unit tests)** للكود.
- تحليل الخوارزميات والتأكد من صحة العمليات الحسابية أو الشروط.

◆ثانياً: طرق تنقيح (Debugging) الأخطاء

١. **استخدام Console**

- console.log()، console.error()، console.warn() لمراقبة القيم ومسار التنفيذ.

٢. **استخدام أدوات المتصفح**

- من خلال **DevTools** في Chrome أو Firefox:
  - التبويب **Sources** يسمح بإضافة **نقاط توقف (breakpoints)**.
  - يمكنك تنفيذ الكود خطوة بخطوة ومراقبة القيم.

٣. **استخدام أدوات مدمجة في المحررات**

- مثل: VS Code Debugger يمكنك تشغيل الكود ومتابعته مباشرة.

٤. **إضافة تنبيهات يدوية**

- عبر alert() أثناء التجربة (لكن غير مفضل في المشاريع الفعلية).

٥. **التحقق من الأخطاء عبر Linting**

- أدوات مثل **ESLint** تنبهك للأخطاء القواعدية والمنطقية الشائعة قبل التنفيذ.

خلاصة سريعة

طريقة المعالجة	النتيجة	التوقيت	نوع الخطأ
تصحيح الكود واستعمال محرر ذكي	توقف البرنامج	قبل التنفيذ	**قواعدي(Syntax)**
استخدام try...catch والتحقق من القيم	توقف مؤقت مع رسالة خطأ	أثناء التنفيذ	**زمن التشغيل (Runtime)**
مراجعة المنطق والاختبار والتنقيح	نتيجة غير صحيحة	بعد التنفيذ	**منطقي(Logical)**

التعامل مع البيانات

- **البيانات:(Data)**
هي الأرقام أو النصوص أو الرموز الخام قبل معالجتها.
مثال"2025-10-10" ,"Ali", 85 :
- **المعلومات:(Information)**
هي ناتج معالجة البيانات بحيث تكتسب معنى مفيداً.
مثال: "درجة الطالب علي هي ٨٥" ـــ هنا تم تفسير البيانات وإعطاؤها معنى.

كيفية تخزين البيانات

- في جافا سكريبت (وفي الحاسوب عمومًا) **كل البيانات تُخزن بصيغة ثنائية (٠ و١)**.
- تختلف طريقة التخزين حسب نوع البيانات:
  - **الأعداد :**تخزن بالصيغة الثنائية.
  - **النصوص :**تخزن باستخدام أنظمة الترميز) مثل ASCII أو.(Unicode
  - **الصور والملفات :**تخزن كسلاسل من البتّات تمثل ألوانًا أو أصواتًا أو قيمًا رقمية.

أنماط تخزين البيانات

١. **الذاكرة المؤقتة:(RAM)**
تخزن البيانات مؤقتًا أثناء تشغيل البرنامج.

٢. **الذاكرة الدائمة (القرص الصلب):**
تخزن البيانات بشكل دائم.

٣. **التخزين السحابي:(Cloud Storage)**
تخزن البيانات عبر الإنترنت على خوادم بعيدة.

ترميز البيانات(Data Encoding)

هي عملية **تحويل النصوص أو الرموز إلى أرقام ثنائية** يمكن للحاسوب فهمها ومعالجتها.

نظامASCII

- **الاسم الكامل***American Standard Code for Information Interchange* **:**
- **الفكرة :**كل حرف أو رمز له رقم ثابت بين ٠ و١٢٧.
مثال:
  - A → 65
  - a → 97
  - 0 → 48

المشاكل:

- لا يدعم إلا **اللغة الإنجليزية** وبعض الرموز البسيطة.
- لا يمكنه تمثيل الحروف العربية أو الرموز العالمية الأخرى.

نظام) Unicode يونيكود(

- **تعريف :**معيار عالمي لترميز جميع لغات العالم في جدول موحّد.
- **آلية العمل :**يعطي **لكل رمز (حرف أو إشارة)** رقمًا فريدًا يسمى **Code Point**مثل:
  - A → U+0041
  - م → U+0645
- **لا يتعارض مع ASCII** لأن أول ١٢٨ رمزاً من يونيكود هي نفسها رموز.ASCII

الميزات:

- يدعم جميع اللغات.

- موحد بين الأنظمة والمنصات.
- يقلل مشاكل اختلاف الترميزات.

🔤 ما هو UTF وما علاقته بيونيكود؟

UTFاختصار لـ **Unicode Transformation Format**،
وهو النظام المسؤول عن **تحويل رموز يونيكود (Unicode code points)** إلى **تسلسل من البِتّات (Bits)** يمكن للحاسوب تخزينه أو نقله عبر الشبكات.

🔷 العلاقة بينهما:

- **يونيكود (Unicode)** هو **الجدول المنطقي** الذي يعرّف كل رمز (مثل حرف أو إشارة) برقم فريد يسمى **Code Point**، مثل:
  - A → U+0041
  - م → U+0645
- **UTF**هو **الطريقة العملية** التي تُحوّل بها هذه الأرقام إلى **تمثيل ثنائي فعلي** ليُخزَّن في الحاسوب.

يونيكود يحدد "ما هو الرمز"، و UTFيحدد "كيف يُخزن هذا الرمز."

⚙️ أنواع UTF والفرق بينها

النوع	عدد البايت لكل رمز	الوصف	المزايا	العيوب
UTF-8	متغير (من ١ إلى ٤ بايت)	يستخدم ١ بايت للحروف الإنجليزية، ويزيد حسب الحاجة للرموز الأخرى.	-موفّر في المساحة -متوافق تمامًا معASCII -الأكثر استخدامًا في الويب	-أطول قليلاً للغات غير اللاتينية
UTF-16	متغير (٢ أو ٤ بايت)	يستخدم ٢ بايت لغالبية الرموز، و٤ بايت للرموز النادرة.	-توازن بين المساحة والسرعة	-ليس متوافقًا تمامًا مع ASCII

	-مناسب للغات الآسيوية			
-يستهلك مساحة كبيرة جدًا	-بسيط وسريع للفك والمعالجة	كل رمز يُخزن في ٤ بايت ثابتة.	ثابت (٤ بايت دائمًا)	**UTF-32**

🔶 **الخلاصة:**

- **Unicode** = نظام ترميز منطقي للأحرف.
- **UTF** = آلية التحويل الفعلي من رموز Unicode إلى بتات.
- **UTF-8** هو الأكثر استخدامًا لأنه يجمع بين الكفاءة والتوافق مع ASCII.

📦 **قواعد البيانات — (Databases) ملخص سريع**

قواعد البيانات هي **طرق منظمة لتخزين البيانات** بحيث يمكن الوصول إليها ومعالجتها بسهولة.

📄 **1.قواعد بيانات الملفات المسطحة(Flat File)**

- عبارة عن **ملفات نصية بسيطة**)مثل CSV أو JSON).
- البيانات تُخزن بشكل تسلسلي بدون علاقات.
- **المزايا :**بسيطة وسهلة الاستخدام.
- **العيوب :**صعبة في البحث والتحديث عند كبر الحجم.

🔗 **2.قواعد البيانات العلائقية(Relational Databases)**

- تعتمد على **الجداول (Tables)** والعلاقات بينها.
- أشهر الأنظمة**MySQL** :، **PostgreSQL**، **SQLite**.
- تستخدم لغة **SQL**للتعامل مع البيانات.
- **المزايا:**
  - منظمة جدًا.
  - تدعم العلاقات والعمليات المعقدة.
- **العيوب :**أقل مرونة مع البيانات غير المنتظمة.

🌐 **3.قواعد البيانات غير العلائقية(NoSQL)**

- تخزن البيانات في **شكل مرن**)مستندات، مفاتيح-قيم، رسومات،...(

- مثال**MongoDB (JSON-like)** :
- **المزايا:**
  - مناسبة للتطبيقات السريعة والمتغيرة.
  - قابلة للتوسع بسهولة.
- **العيوب:**
  - لا تدعم العلاقات المعقدة بشكل مباشر.
  - لا تستخدم SQL القياسية.