Typescript

استاد درس: دکتر آرش شفیعی

پدید آورندگان: محمدامین صابری - ۹۹۳۶۲۳۰۲۶، ابوالفضل شیشه‌گر – ۹۹۳۶۲۳۰۲۵

فاز دوم

دانشکده مهندسی کامپیوتر

دانشگاه اصفهان

زمستان ۱۴۰۲

فهرست عناوین

امکانات زبان برنامه نویسی TypeScript

زبان برنامه نویسی TypeScript، یک زبان کامپایل شده از جاوااسکریپت است که برنامه نویسی تابعی را پشتیبانی می‌کند. TypeScript، به توسعه دهندگان اجازه ایجاد اپلیکیشن‌های بزرگ و پیچیده را می‌دهد. این زبان ویژگی‌ها و مفاهیمی را از زبان‌هایی مانند C# و Java نیز به ارث برده که باعث شده انضباط و دستورات بیشتری نسبت به جاوااسکریپت خالی پیدا کند.

در TypeScript، شما می‌توانید سازوکارهای برنامه‌نویسی تابعی را پیاده‌سازی کنید. این امکانات بر اساس قابلیت‌های جاوااسکریپت موجود هستند و با استفاده از تایپ‌ها و ویژگی‌های اضافی TypeScript قابل بهبود است. در زیر، چند مثال از سازوکارهای برنامه‌نویسی تابعی در TypeScript آورده شده است:

۱. توابع لامبدا:

توابع لامبدا یا توابع بی‌نام می‌توانند به صورت مستقیم در TypeScript تعریف شوند:

```
تعریف تابع لامبدا برای جمع دو عدد//
const add = (x: number, y: number): number => x + y;

استفاده از تابع لامبدا//
const result = add(5, 3);
console.log(result);// خروجی : ۸
```

۲. ارسال تابع به تابع:

می‌توانید توابع را به توابع دیگر پاس داده و از آن‌ها به عنوان پارامتر استفاده کنید:

```
تعریف تابع با پارامتر تابع//
function operate(x: number, y: number, operation: (a: number, b: number) => number): number {
  return operation(x, y);
```

```
}
```

استفاده از تابع با ارسال تابع لامبدا به عنوان پارامتر//

```
const result = operate(5, 3, (a, b) => a + b);
console.log(result);//۸ : خروجی
```

۳. بازگرداندن تابع از تابع:

توابع می‌توانند توابع دیگر را بازگردانند:

تعریف تابع که تابع لامبدا را بازگردانی می‌کند//

```
function createAdder(y: number): (x: number) => number}
  return (x: number) => x + y;
{
```

استفاده از تابع بازگردانده شده//

```
const addFive = createAdder;(۵)
const result = addFive;(۳)
console.log(result)۸ // خروجی: ;
```

۴. توابع نگاشت، فیلتر، کاهش و غیره:

از توابع بالا به همراه توابع نگاشت (map)، فیلتر (filter) و کاهش (reduce) می‌توانید برای کار با آرایه‌ها و داده‌ساختارهای دیگر استفاده کنید:

تعریف یک آرایه از اعداد//

```
const numbers = [1, 2, 3, 4, 5];
```

استفاده از توابع نگاشت و فیلتر//

```
const squaredNumbers = numbers.map((x) => x ** 2);
const evenNumbers = numbers.filter((x) => x % 2 === 0);
```

استفاده از تابع کاهش برای جمع اعداد//

```
const sum = numbers.reduce((acc, curr) => acc + curr, 0);
```

```
console.log(squaredNumbers)// [25, 16, 9, 4, 1]
```
خروجی :

```
console.log(evenNumbers);   // [4, 2]
```
خروجی :

```
console.log(sum)            //
```
خروجی : ۱۵

این مثال‌ها نشان می‌دهند که TypeScript به شما امکان پیاده‌سازی سازوکارهای برنامه‌نویسی تابعی را با استفاده از ویژگی‌ها و تایپ‌های خود می‌دهد.

استفاده از توابع نگاشت و فیلتر و کاهش، به دلیل کاربردی بودن و قابلیت استفاده در بسیاری از موارد، می‌تواند به بهبود کارایی برنامه کمک کند. این توابع در TypeScript نیز پیاده‌سازی شده‌اند.

توابع نگاشت، توابعی هستند که یک تابع را به یک مجموعه داده اعمال می‌کنند و نتیجه را به عنوان یک مجموعه داده جدید باز می‌گردانند. این توابع، به دلیل کاربردی بودن و قابلیت استفاده در بسیاری از موارد، می‌توانند به بهبود کارایی برنامه کمک کنند.

تعریف یک آرایه از اعداد//

```
const numbers = [1, 2, 3, 4, 5];
```

توابع نگاشت: مربع کردن اعداد//

```
const squaredNumbers = numbers.map((x) => x ** 2);
console.log(squaredNumbers); // [25, 16, 9, 4, 1]
```
خروجی :

توابع فیلتر، توابعی هستند که یک تابع را به یک مجموعه داده اعمال می‌کنند و تمام مقادیری را که شرط مشخص شده را برآورده نمی‌کنند، حذف می‌کنند. این توابع، به دلیل کاربردی بودن و قابلیت استفاده در بسیاری از موارد، می‌توانند به بهبود کارایی برنامه کمک کنند.

تعریف یک آرایه از اعداد//

```
const numbers  = [1, 2, 3, 4, 5];
```

تابع فیلتر: انتخاب اعداد زوج//

```
const evenNumbers = numbers.filter((x) => x % 2 === 0);
console.log(evenNumbers); // [4, 2] خروجی :
```

توابع کاهش، توابعی هستند که یک تابع را به یک مجموعه داده اعمال می‌کنند و نتیجه را به عنوان یک مقدار باز می‌گردانند. این توابع، به دلیل کاربردی بودن و قابلیت استفاده در بسیاری از موارد، می‌توانند به بهبود کارایی برنامه کمک کنند.

تعریف یک آرایه از اعداد//

```
const numbers = [1, 2, 3, 4, 5];
```

تابع کاهش: جمع اعداد//

```
const sum = numbers.reduce((acc, curr) => acc + curr, 0);
console.log(sum); // ۱۵ خروجی:
```

برای مقایسه سرعت اجرای توابع برنامه‌نویسی توابعی با حلقه تکرار، می‌توانیم از بسته `performance-now` استفاده کنیم. در اینجا، یک مثال ساده با استفاده از حلقه تکرار و توابع فیلتر و نگاشت ارائه شده است:

```
import { performance } from 'perf_hooks;'
```

تعریف یک آرایه بزرگ از اعداد//

```
const largeArray = Array.from({ length: 1000000 }, (_, index) => index
+ 1);
```

استفاده از حلقه تکرار//

```
const startTimeLoop = performance.now();
```

```
let sumLoop = 0;
for (let i = 0; i < largeArray.length; i++){
 if (largeArray[i] % 2 === 0)}
  sumLoop += largeArray[i] ** 2;
 }
}

const endTimeLoop = performance.now;()
console.log('حلقه تکرار: ' ,sumLoop);
console.log('زمان اجرا با حلقه تکرار: ', endTimeLoop – startTimeLoop, 'میلی ثانیه');

//استفاده از توابع نگاشت و فیلتر
const startTimeFunctional = performance.now();
const sumFunctional = largeArray
. filter((x) => x % 2 === 0)
. map((x) => x ** 2)
. reduce((acc, curr) => acc + curr, 0);
const endTimeFunctional = performance.now();
console.log('برنامه نویسی تابعی: ', sumFunctional);
console.log('زمان اجرا با برنامه نویسی تابعی: ', endTimeFunctional -
startTimeFunctional, 'میلی ثانیه');
```

در این مثال، یک آرایه بزرگ از اعداد تولید شده و سپس با استفاده از یک حلقه تکرار و توابع فیلتر و نگاشت مقایسه شده‌اند. ممکن است در مواقعی نتایج متفاوت باشند، اما عموماً توابع برنامه‌نویسی تابعی به دلیل بهینه‌سازی‌های موجود در جاوااسکریپت بهترین عملکرد را ارائه کنند.

بله، زبان برنامه نویسی TypeScript برنامه نویسی رویه‌ای را پشتیبانی می‌کند. TypeScript یک زبان برنامه‌نویسی چندسکویی، متن باز و کامپایلری است که توسط شرکت مایکروسافت توسعه داده شده و پشتیبانی می‌شود. TypeScript از تعریف نوع برای کتابخانه‌های جاوا اسکریپت پشتیبانی می‌کند و از برنامه نویسی شیء گرا (OOP) و مفاهیم آن (مانند رابط، کلاس، ارث بری و غیره) پشتیبانی می‌کند.

زبان برنامه‌نویسی TypeScript به صورت اساسی یک زبان برنامه‌نویسی شیءگرا است که بر پایه JavaScript توسعه یافته است. اما TypeScript همچنین از ویژگی‌های برنامه‌نویسی رویه‌ای (Type System) پیشرفته‌تری نسبت به JavaScript پشتیبانی می‌کند. این ویژگی‌ها اجازه می‌دهند تا توسعه‌دهندگان بهترین استفاده از تایپ‌ها و پیشرفت‌های برنامه‌نویسی رویه‌ای که در TypeScript موجود است، ببرند.

با استفاده از TypeScript، می‌توانید تایپ‌های متغیرها، پارامترها، خروجی توابع و سایر عناصر کد را به صورت صریح تعیین کنید. این کار باعث افزایش قابلیت خوانایی کد، افزایش امنیت و کاهش احتمال خطاها می‌شود. TypeScript همچنین از ویژگی‌های دیگر برنامه‌نویسی رویه‌ای مانند ارث‌بری، اینترفیس، جنریک، و متغیرهای ثابت پشتیبانی می‌کند.

برای استفاده از TypeScript، شما کد خود را با فرمت `.ts` ذخیره می‌کنید و سپس با استفاده از کامپایلر TypeScript به کد JavaScript تبدیل می‌شود. این کد JavaScript حاوی تمام تغییرات و توابع زبان TypeScript است و می‌تواند بر روی هر محیطی که JavaScript را اجرا می‌کند، اجرا شود.

بنابراین، می‌توان گفت که TypeScript همچنان به صورت اساسی یک زبان برنامه‌نویسی شیءگرا است، اما با افزودن ویژگی‌های برنامه‌نویسی رویه‌ای، از توسعه ساختار یافته‌تر و امن‌تر کد پشتیبانی می‌کند.

در زیر، به توضیح برخی از مفاهیم و الگوهای مرتبط با زیربرنامه‌ها، ارسال متغیرها به توابع، برنامه‌نویسی عمومی و غیره در TypeScript می‌پردازم:

۱. زیربرنامه‌ها (Subprograms):

زیربرنامه‌ها به توابع یا متدهای جداگانه اشاره دارند که در داخل یک برنامه یا نرم‌افزار وظیفه‌های خاصی را انجام می‌دهند. این مفهوم به توسعه‌دهندگان این امکان را می‌دهد تا برنامه‌ها را به بخش‌های کوچکتر و قابل مدیریت‌تر تقسیم کنند.

زیربرنامه یا تابع محاسبه مساحت دایره//

```
function calculateCircleArea(radius: number): number {
  return Math.PI * radius ** 2;
```

۲. ارسال متغیرها به توابع:

در TypeScript، متغیرها به توابع می‌توانند به صورت زیر ارسال شوند:

ارسال به صورت مستقیم:

```
function add(x: number, y: number): number {
  return x + y;
}

const result = add(3, 5);
```

ارسال به صورت شیء:

```
interface Point {
  x: number;
  y: number;
}

function calculateDistance(point: Point): number {
  return Math.sqrt(point.x ** 2 + point.y ** 2);
}

const distance = calculateDistance({ x: 3, y: 4 });
```

۳. برنامه‌نویسی عمومی (Generic Programming) :

برنامه‌نویسی عمومی به شیوه‌ای اشاره دارد که توابع و کلاس‌ها به صورت کلی برای نوع‌های مختلف داده قابل استفاده باشند. این امکان با استفاده از جنریک‌ها در TypeScript وارد می‌شود.

```
تابع عمومی برای جمع دو مقدار//
function add<T>(x: T, y: T): T {
   return x + y;
}

const result = add(5, 3); // number نوع :
const concatenated = add("Hello", " World");// string نوع :
```

۴. متغیرهای تایپ (Type Variables):

متغیرهای تایپ یا تایپ‌های جدید می‌توانند برای ایجاد نوع‌های سفارشی و کلیات در TypeScript استفاده شوند.

```
ایجاد تایپ جدید به عنوان مثال//
type Point = {
  x: number;
  y: number;
};

استفاده از تایپ در تابع//
function calculateDistance(point: Point): number {
  return Math.sqrt(point.x ** 2 + point.y ** 2);
```

```
}
```

۵. نوع داده‌ها و خروجی توابع :

استفاده از توابع با تعیین نوع داده و خروجی آنها افزایش امنیت و خوانایی کد را بهبود می‌بخشد.

```
تعیین نوع داده و خروجی تابع//
function divide(x: number, y: number): number {
  if (y !== 0) {
    return x / y;
  } else {
    throw new Error("Cannot divide by zero.");
  }
}

const result = divide(10, 2);
```

در کل، TypeScript امکانات بسیاری را برای ساختاردهی بهتر کد و بهبود امنیت و کارایی ارائه می‌دهد که می‌تواند در توسعه برنامه‌ها و زیربرنامه‌ها به شدت مفید باشد.

TypeScript به صورت کامل از برنامه‌نویسی شیءگرا پشتیبانی می‌کند. در واقع، TypeScript از سینتکس و ویژگی‌های برنامه‌نویسی شیءگرایی JavaScript بهره می‌برد و علاوه بر آن، ویژگی‌هایی را افزوده است که قابلیت‌ها و امکانات برنامه‌نویسی شیءگرا را ارتقاء می‌دهد.

۱. ساختارهای موجود در TypeScript:

- الگوهای داده:

رابط (Interface): یک الگوی داده برای تعیین ساختار شیء است.

```
interface Person {
  name: string;
  age: number;
```

```
}
```

- کلاس‌ها:

کلاس (Class): یک قالب برنامه‌نویسی شیءگرا که این امکان را فراهم می‌کند تا یک شیء از آن ساخته شود.

```
class Animal {
  name: string;

  constructor(name: string) {
    this.name = name;
  }

  makeSound(): void {
    console.log("Some generic sound");
 }
}
```

- توابع عضو کلاس:

متد (Method): یک تابع که به یک شیء خاص مرتبط است.

```
class Animal {
//(قسمت‌های دیگر کلاس) ...

  makeSound(): void {
    console.log("Some generic sound");
  }
}
```

۲. مفاهیم برنامه‌نویسی شیءگرا در TypeScript:

- حافظه:

نمونه (Instance): یک شیء از یک کلاس که می‌تواند در حافظه ساخته شود.

```
const cat = new Animal("Fluffy");
```

- چندریختی:

وراثت (Inheritance): امکان ارث‌بری از یک کلاس به کلاس دیگر.

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

- پلی‌مورفیسم:

پلی‌مورفیسم (Polymorphism): امکان اجرای متدها به شکل متفاوت در کلاس‌های مختلف.

```
const animals: Animal[] = [new Animal("Fluffy"), new Dog("Buddy")];

for (const animal of animals) {
  animal.makeSound;()
}
```

- انکپسولیشن:

انکپسولیشن (Encapsulation): محدود کردن دسترسی به ویژگی‌ها و متدهای یک شیء.

```
class BankAccount {
  private balance: number;
  constructor(initialBalance: number) {
    this.balance = initialBalance;
  }

  getBalance(): number {
    return this.balance;
  }

  deposit(amount: number): void {
    this.balance += amount;
  }

  withdraw(amount: number): void {
    if (amount <= this.balance)}
      this.balance -= amount;
    } else {
      console.log("Insufficient funds!");
    }
  }
}
```

این مثال‌ها نشان می‌دهند چگونه TypeScript از مفاهیم برنامه‌نویسی شیءگرا استفاده می‌کند و امکاناتی مانند ارث‌بری، پلی‌مورفیسم، انکپسولیشن، و غیره را فراهم می‌کند. با استفاده از این مفاهیم، می‌توانید کد مقیاس‌پذیر، خوانا و قابل نگهداری تولید کنید.

زبان برنامه نویسی TypeScript از مفاهیم همروندی پشتیبانی می کند. برای پیاده سازی همروندی در TypeScript، می توانید از سازوکارهای موجودی مانند سمافورها، قفل ها، مکانیزم های ارسال پیام و ریسه ها استفاده کنید.

سمافورها: یک سمافور، یک متغیر صحیح است که به عنوان یک نشانگر برای دسترسی به منابع مشترک در برنامه های همروند استفاده می شود. سمافورها به برنامه های همروند کمک می کنند تا به منابع مشترک دسترسی داشته باشند و در عین حال از تداخل همروندی جلوگیری کنند.

قفل ها: قفل ها به برنامه های همروند کمک می کنند تا به منابع مشترک دسترسی داشته باشند و در عین حال از تداخل همروندی جلوگیری کنند. قفل ها به برنامه های همروند کمک می کنند تا به منابع مشترک دسترسی داشته باشند و در عین حال از تداخل همروندی جلوگیری کنند.

مکانیزم های ارسال پیام: مکانیزم های ارسال پیام به برنامه های همروند کمک می کنند تا اطلاعات را بین ریسه های مختلف به اشتراک بگذارند. این مکانیزم ها شامل صف ها، کانال ها و پردازشگرهای پیام می شوند.

ریسه ها: ریسه ها به برنامه های همروند کمک می کنند تا به صورت همزمان اجرا شوند. TypeScript از ریسه های وب استفاده می کند که به برنامه های همروند کمک می کند تا به صورت همزمان اجرا شوند.

مقایسه زبان یک الگوریتم در زبان انتخابی با یک زبان سطح بالاتر و پایین تر

مقایسه الگوریتم در زبان‌های مختلف _ محمدامین صابری

الگوریتمی را که در اینجا پیاده سازی می‌شود، الگوریتم مرتب‌سازی میانه‌ای (Merge Sort) است. این الگوریتم برای مقایسه اندازه کد و زمان اجرا، به خوبی مناسب است.

الگوریتم مرتب‌سازی میانه‌ای (Merge Sort) در TypeScript:

```
// TypeScript Implementation of Merge Sort Algorithm

mergeSort(arr: number[]): number} []
  if (arr.length <= 1) {
    return arr;
  }

  const mid = Math.floor(arr.length / 2);
  const left = arr.slice(0, mid);
  const right = arr.slice(mid);

  return merge(mergeSort(left), mergeSort(right));
}

merge(left: number[], right: number[]): number} []
  let result: number[]=[];
```

```
  let leftIndex = 0;
  let rightIndex = 0;

  while (leftIndex < left.length && rightIndex < right.length){
    if (left[leftIndex] < right[rightIndex]) {
      result.push(left[leftIndex]);
      leftIndex;++
    } else{
      result.push(right[rightIndex]);
      rightIndex;++
    }
  }

  return
result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
}

//Example usage:
const unsortedArray = [64, 34, 25, 12, 22, 11, 90];
const sortedArray = mergeSort(unsortedArray);
console.log("Sorted Array:", sortedArray);
```

مقایسه با یک زبان سطح پایین‌تر (مثلاً C):

الگوریتم مرتب‌سازی میانه‌ای (Merge Sort) در C:

```
// C Implementation of Merge Sort Algorithm


#include <stdio.h>


void merge(int arr[], int left[], int leftSize, int right[], int
rightSize){
  int i = 0, j = 0, k = 0;


  while (i < leftSize && j < rightSize) {
    if (left[i] <= right[j])}
      arr[k] = left[i];
      i;++
    } else {
      arr[k] = right[j];
      j;++
    }
    K++;
  }
```

```
  while (i < leftSize) {
    arr[k] = left[i];
    i++;
    k++;
  }

  while (j < rightSize) {
    arr[k] = right[j];
    j++;
    k++;
  }
}

void mergeSort(int arr[], int size) {
  if (size > 1) {
    int mid = size / 2;
    int left[mid];
    int right[size - mid];

    for (int i = 0; i < mid; i++) {
      left[i] = arr[i];
    }

    for (int i = mid; i < size; i++) {
      right[i - mid] = arr[i];
    }
```

```
    mergeSort(left, mid);
    mergeSort(right, size - mid);

    merge(arr, left, mid, right, size - mid);
  }
}

//Example usage:
int main(){
  int unsortedArray[] = {64, 34, 25, 12, 22, 11, 90};
  int size = sizeof(unsortedArray) / sizeof(unsortedArray[0]);

  mergeSort(unsortedArray, size);

  printf("Sorted Array: ");
  for (int i = 0; i < size; i++) {
    printf("%d ", unsortedArray[i]);
  }

  return 0;
}
```

مقایسه:

۱. زمان اجرا:

- الگوریتم مرتب‌سازی میانه‌ای در هر دو زبان در حالت متوسط دارای زمان اجرای O(n log n) است. بنابراین، این نکته در این مقایسه مهم است که زمان اجرا تقریباً یکسان خواهد بود.

۲. اندازه کد:

- الگوریتم مرتب‌سازی میانه‌ای در TypeScript به دلیل خصوصیات زبان و انعطاف بیشتری که از جمله دینامیک بودن نوع داده‌ها و ابزارهای متداول مانند توابع بازگشتی در زبان دارد، ممکن است کد کمتری داشته باشد.
- الگوریتم مشابه در C کمی بیشترین کد نوشته شده و نیاز به مدیریت دقیق‌تر حافظه دارد.

در نهایت، انتخاب زبان برنامه‌نویسی بیشتر به ویژگی‌ها و نیازهای پروژه، تیم توسعه، و دیگر عوامل بستگی دارد. TypeScript برای پروژه‌های وب و ساختارهای بزرگتر با انعطاف بیشتر و امکانات نوع داده جدید مناسب است، در حالی که C برای پروژه‌های سیستم و کارهایی که به بهینه‌سازی زمان اجرا نیاز دارند، مناسب‌تر است.

به عنوان یک زبان سطح بالاتر از TypeScript، می‌توانیم Python را در نظر بگیریم. در اینجا، الگوریتم مرتب‌سازی میانه‌ای (Merge Sort) را در Python پیاده‌سازی می‌کنم و با TypeScript مقایسه می‌کنم:

الگوریتم مرتب‌سازی میانه‌ای (Merge Sort) در Python:

```
#Python Implementation of Merge Sort Algorithm

def merge_sort(arr):
  if len(arr) <= 1:
    return arr

  mid = len(arr) // 2
  left = arr[:mid]
  right = arr[mid:]

  return merge(merge_sort(left), merge_sort(right))

def merge(left, right):
  result = []
  left_index = right_index = 0

  while left_index < len(left) and right_index < len(right):
    if left[left_index] < right[right_index]:
      result.append(left[left_index])
      left_index += 1
    else:
      result.append(right[right_index])
      right_index += 1
```

```
  result.extend(left[left_index:])
  result.extend(right[right_index:])
  return result


#Example usage:
unsorted_array = [64, 34, 25, 12, 22, 11, 90]
sorted_array = merge_sort(unsorted_array)
print("Sorted Array:", sorted_array)
```

مقایسه با TypeScript:

۱. زمان اجرا:

- زمان اجرای الگوریتم مرتب‌سازی میانه‌ای در Python و TypeScript برابر است و به همان اندازه O(n log n) است.

۲. اندازه کد:

- کد Python به دلیل ساختار ساده‌تر زبان و نیاز کمتر به تعریف نوع داده‌ها معمولاً کمتر و خواناتر است.
- کد TypeScript با اضافه کردن اطلاعات نوع، توابع بازگشتی و سایر ویژگی‌های زبان، ممکن است بیشتر باشد.

۳. امکانات زبان:

- TypeScript از نظر امکانات زبانی و ابزارهای نوع داده برنامه‌نویس را بهتر پشتیبانی می‌کند.
- Python با انعطاف بالایی در انجام تسک‌های سریع و توسعه سریع، برای پروژه‌هایی که انعطاف و خوانایی بالاتر مهم است، ممکن است مناسب‌تر باشد.

در نهایت، انتخاب بین TypeScript و Python نیز بستگی به نیازها، محیط پروژه، و ترجیحات تیم توسعه دارد. هرکدام از این زبان‌ها ویژگی‌ها و مزایا خود را دارند و باید با توجه به موارد مختلف انتخاب شوند.

الگوریتمی که در اینجا پیاده‌سازی می‌کنم، الگوریتم جستجوی دودویی (Binary Search) است. این الگوریتم معمولاً برای جستجو در یک آرایه مرتب‌شده استفاده می‌شود.

الگوریتم جستجوی دودویی (Binary Search) در TypeScript:

```
//TypeScript Implementation of Binary Search Algorithm


binarySearch(arr: number[], target: number): number}
  let left = 0;
  let right = arr.length - 1;


  while (left <= right)}
    const mid = Math.floor((left + right) / 2);


    if (arr[mid] === target){
      return mid; // پیدا شد!
    }else if (arr[mid] < target){
      left = mid + 1;
    }else{
      right = mid - 1;
    }
  }
```

```
  return -1; // نتیجه پیدا نشد !
}


//Example usage:
const sortedArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const targetValue = 7;
const result = binarySearch(sortedArray, targetValue);

if (result !== -1) {
  console.log(`Target ${targetValue} found at index ${result}.`);
} else {
  console.log(`Target ${targetValue} not found in the array.`);
}
```

الگوریتم جستجوی دودویی (Binary Search) در Python:

```
#Python Implementation of Binary Search Algorithm

def binary_search(arr, target):
  left, right = 0, len(arr) - 1

  while left <= right:
    mid = (left + right) // 2

    if arr[mid] == target:
      return mid # پیدا شد !
    elif arr[mid] < target:
      left = mid + 1
    else:
      right = mid - 1

  return -1 # نتیجه پیدا نشد .

#Example usage:
sorted_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
target_value = 7
result = binary_search(sorted_array, target_value)
```

```
if result != -1:
  print(f"Target {target_value} found at index {result}.")
else:
  print(f"Target {target_value} not found in the array.")
```

الگوریتم جستجوی دودویی (Binary Search) در C:

```
//C Implementation of Binary Search Algorithm

#include <stdio.h>

int binary_search(int arr[], int target, int size){
  int left = 0;
  int right = size - 1;

  while (left <= right){
    int mid = (left + right) / 2;

    if (arr[mid] == target){
      return mid//پیدا شد!;
    }else if (arr[mid] < target){
      left = mid + 1;
    }else{
      right = mid - 1;
```

```
    }
  }

  return -1; // نتیجه پیدا نشد
}

//Example usage:
int main(){
  int sorted_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int target_value = 7;
  int size = sizeof(sorted_array) / sizeof(sorted_array[0]);

  int result = binary_search(sorted_array, target_value, size);

  if (result != -1){
    printf("Target %d found at index %d.\n", target_value, result);
  }else{
    printf("Target %d not found in the array.\n", target_value);
  }

  return 0;
}
```

مقایسه:

۱. زمان اجرا:

- زمان اجرای الگوریتم جستجوی دودویی در هر سه زبان به طور معمول O(log n) است و بنابراین به صورت مشابه خواهد بود.

۲. اندازه کد:

- کد Python معمولاً کمتر و خواناتر است به دلیل ساختار ساده‌تر زبان.
- کد TypeScript بیشتر از Python با توجه به ویژگی‌های زبان و ابزارهای متداولی که دارد.

۳. امکانات زبان:

- TypeScript از نظر امکانات زبانی و نوع داده برنامه‌نویس را بهتر پشتیبانی می‌کند.
- Python با انعطاف بالایی در انجام تسک‌های سریع و توسعه سریع، برای پروژه‌هایی که انعطاف و خوانایی بالاتر مهم است، ممکن است مناسب‌تر باشد.
- C نیاز به مدیریت دقیق‌تر حافظه دارد و برای پروژه‌هایی که به بهینه‌سازی زمان اجرا نیاز دارند، مناسب‌تر است.

همانند مقایسه‌ی قبلی، انتخاب بین این زبان‌ها بستگی به نیازها و شرایط پروژه دارد.

منابع و ماخذ

۱. [1]: [آموزش Typescript در ۳۰ دقیقه - راکت](https://roocket.ir/articles/learn-typescript-in-30-minutes)

۲. [2]: [آموزش تایپ اسکریپت (TypeScript)](https://academyit.net/products/typescript).

۱. آموزش Typescript در ۳۰ دقیقه - راکت. https://roocket.ir/articles/learn-typescript-in-30-minutes.

۲. مروری بر زبان برنامه نویسی TypeScript – آموزش TypeScript. https://sourcesara.com/typescript-overview/.

۳. آموزش تایپ اسکریپت (TypeScript). https://academyit.net/products/typescript.

۴. تایپ اسکریپت (TypeScript) چیست و چرا باید آن را یاد بگیریم؟. https://7learn.com/blog/what-is-typescript.

۵. en.wikipedia.org. https://en.wikipedia.org/wiki/TypeScript

۶. [1]: [آموزش Typescript در ۳۰ دقیقه - راکت](https://roocket.ir/articles/learn-typescript-in-30-minutes)

۷. [2]: [آموزش تایپ اسکریپت (TypeScript)](https://academyit.net/products/typescript).

۸. نکات و ترفندهای تایپ اسکریپت — راهنمای کاربردی – فرادرس - مجله. https://blog.faradars.org/typescript-tips-and-tricks/.

۹. مفهوم تابع در تایپ اسکریپت — به زبان ساده – فرادرس - مجله. https://blog.faradars.org/functions-in-typescript/.

۱۰. انواع تابع-پارامتر ژنریک در تایپ اسکریپت — به زبان ساده. https://blog.faradars.org/typescript-generic-objects/.

۱۱. راهنمای جامع تایپ اسکریپت (Typescript) — از صفر تا صد. https://blog.faradars.org/a-crash-course-in-typescript/.

۱۲. رویه های مناسب کدنویسی تایپ اسکریپت | راهنمای کاربردی. https://blog.faradars.org/4-ways-to-write-more-effective-typescript/.

۱۳. مروری بر زبان برنامه نویسی TypeScript – آموزش TypeScript. https://sourcesara.com/typescript-overview/.

۱۴. تایپ اسکریپت (Type Script)چیست؟ + کاربردهای آن. https://codeyad.com/mag/post/what-is-typescript-uses-of-typescript.

۱۵.راهنمای جامع تایپ اسکریپت (Typescript) — از صفر تا صد. https://blog.faradars.org/a-crash-course-in-typescript/.
۱۶.تایپ اسکریپت (TypeScript) چیست و چرا باید آن را یاد بگیریم؟. https://7learn.com/blog/what-is-typescript.
۱۷.https://en.wikipedia.org/wiki/TypeScript. en.wikipedia.org.
۱۸.ارسال آرایه ها به توابع در زبان برنامه نویسی TypeScript - آموزش ....
https://sourcesara.com/typescript-passing-arrays-to-functions/.
۱۹.روش ارسال پارامترها به توابع | آموزش برنامه نویسی.
https://sourceiran.com/articles/%D8%B1%D9%88%D8%B4-%D8%A7%D8%B1%D8%B3%D8%A7%D9%84-%D9%BE%D8%A7%D8%B1%D8%A7%D9%85%D8%AA%D8%B1%D9%87%D8%A7-%D8%A8%D9%87-%D8%AA%D9%88%D8%A7%D8%A8%D8%B9/.
۲۰.متغیرها در زبان برنامه نویسی TypeScript – آموزش TypeScript. https://sourcesara.com/typescript-variables/.
۲۱.راهنمای جامع تایپ اسکریپت (Typescript) — از صفر تا صد. https://blog.faradars.org/a-crash-course-in-typescript/.
۲۲.: [مروری بر زبان برنامه نویسی TypeScript – آموزش TypeScript](https://sourcesara.com/typescript-overview/)
۲۳.: [TypeScript - Wikipedia](https://en.wikipedia.org/wiki/TypeScript)
۲۴.: [TypeScript: JavaScript Development at Application Scale](https://www.typescriptlang.org/).
۲۵.مروری بر زبان برنامه نویسی TypeScript – آموزش TypeScript. https://sourcesara.com/typescript-overview/.
۲۶.آموزش زبان برنامه نویسی TypeScript - سورس سرا. https://sourcesara.com/typescript-tutorial/.
۲۷.قواعد نحوی دستورات زبان برنامه نویسی TypeScript - آموزش .... TypeScript
https://sourcesara.com/typescript-basic-syntax/.
۲۸.تایپ اسکریپت (Type Script)چیست؟ + کاربردهای آن. https://codeyad.com/mag/post/what-is-typescript-uses-of-typescript.
۲۹.https://en.wikipedia.org/wiki/TypeScript. en.wikipedia.org.
۳۰.دانلود دو جزوه برنامه نویسی همروند (Concurrent) - نواندیشان.
https://noandishaan.com/59651/%d8%ac%d8%b2%d9%88%d9%87-

%d8%a8%d8%b1%d9%86%d8%a7%d9%85%d9%87-
%d9%86%d9%88%db%8c%d8%b3%db%8c-
./%d9%87%d9%85%d8%b1%d9%88%d9%86%d8%af

۳۱.سمافور چیست ؟ – در سیستم عامل و به زبان ساده – فرادرس - مجله.

https://blog.faradars.org/%d8%b3%d9%85%d8%a7%d9%81%d9%88%d8%b1-
./%da%86%db%8c%d8%b3%d8%aa

۳۲.کنترل همروندی - ویکی‌پدیا، دانشنامه آزاد.

https://fa.wikipedia.org/wiki/%DA%A9%D9%86%D8%AA%D8%B1%D9%84_%D9%87%D
.9%85%D8%B1%D9%88%D9%86%D8%AF%DB%8C