Duplicate of Auto Test Tutorial

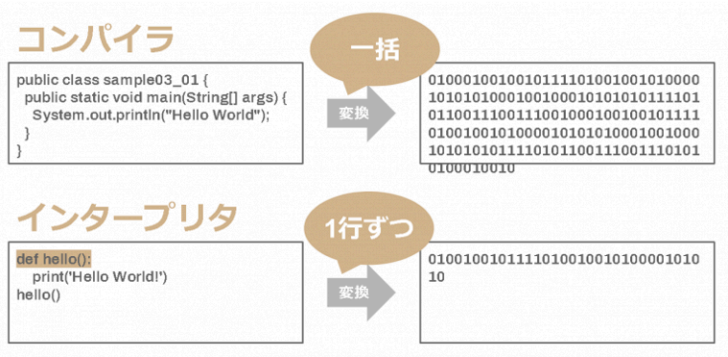
I. プログラミング言語

1. コンパイル (Compile)

- 高水準言語(プログラミング言語)で書かれたプログラムをコンピュータが理解可能な形に変換する(機械語)処理を行う言語プロセッサ
-

2. コンパイル言語 vs. スクリプト言語 (Compiled Language vs. Scripting Language)

コンパイル言語は「高速で安定した動作」スクリプト言語は「手軽に素早く動かす」を目指す傾向があります。



	コンパイル言語 (Compiled Language)	スクリプト/インタプリタ言語 （Scripting/ Interpreting Language）
コンセプト	ソースコードを事前に全て機械語に変換（コンパイル）し、実行ファイルを作成する。	ソースコードを一行ずつ解釈しながら実行する（インタプリタ方式）。
メリット	• 実行速度が速い • メモリ効率が良い • 大規模開発に向き	• 開発が手軽で簡単 • 学習コストが低い • コード修正後すぐに実行確認可
デメリット	• コンパイルの手間がかかる • デバッグに時間がかかる場合があ	• コンパイル言語より実行速度が遅い • メモリ消費が多い傾向。
例	C, C++, C#, Java, Go, Rust	Python, Ruby, JavaScript, PHP, Perl, Bash

3. それぞれのプログラミング言語

1) Print out; Hello, World!

a. C++

```cpp
#include <iostream>

using namespace std; // Using the standard namespace

int main() {
    cout << "Hello, World!";
    return 0;
}
```

b. Java

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!"); // Prints the message to the console
    }
}
```

c. JavaScript

```javascript
console.log('Hello, World!');
```

d. Python

```python
print("Hello, World!")
```

2) オブジェクト指向プログラミング (Object Oriented Programming; OOP)

a. C++

```cpp
#include <iostream>
#include <string>
using namespace std;

// Class definition
class Car {
private:
    string brand;  // property
    string model;  // property

public:
    // Constructor
    Car(string b, string m) {
        brand = b;
        model = m;
    }

    // Method
    void showDetails() {
        cout << "This car is a " << brand << " " << model << "." << endl;
    }
};

int main() {
    // Creating objects (instances) from the class
    Car car1("Toyota", "Corolla");
    Car car2("Honda", "Civic");

    // Using the objects
    car1.showDetails(); // Output: This car is a Toyota Corolla.
    car2.showDetails(); // Output: This car is a Honda Civic.

    return 0;
}
```

b. Java

```java
// Class definition
class Car {
    // Properties
    private String brand;
    private String model;

    // Constructor
    public Car(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }
```

```java

    // Method
    public void showDetails() {
        System.out.println("This car is a " + brand + " " + model + ".");
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating objects (instances) from the class
        Car car1 = new Car("Toyota", "Corolla");
        Car car2 = new Car("Honda", "Civic");

        // Using the objects
        car1.showDetails(); // Output: This car is a Toyota Corolla.
        car2.showDetails(); // Output: This car is a Honda Civic.
    }
}
```

c. JavaScript

```javascript
// Class definition
class Car {
  constructor(brand, model) {
    this.brand = brand; // property
    this.model = model; // property
  }

  // Method
  showDetails() {
    console.log(`This car is a ${this.brand} ${this.model}.`);
  }
}

// Creating objects (instances) from the class
const car1 = new Car("Toyota", "Corolla");
const car2 = new Car("Honda", "Civic");

// Using the objects
car1.showDetails(); // Output: This car is a Toyota Corolla.
car2.showDetails(); // Output: This car is a Honda Civic.
```

d. Python

```python
# Class definition
class Car:
    # Constructor
    def __init__(self, brand, model):
        self.brand = brand   # property
        self.model = model   # property

    # Method
    def show_details(self):
        print(f"This car is a {self.brand} {self.model}.")

# Creating objects (instances) from the class
car1 = Car("Toyota", "Corolla")
car2 = Car("Honda", "Civic")

# Using the objects
car1.show_details()  # Output: This car is a Toyota Corolla.
car2.show_details()  # Output: This car is a Honda Civic.
```

5. 練習問題

🖌️ Auto Test Tutorial quiz1.pdf

Ⅱ. JavaScript

1. JavaScriptとは？

1995年にブランドン・アイクが開発し、1997年にECMA International所属のTC39で標準化されたプロトタイプベースのプログラミング言語で、スクリプト言語に該当する。
特別な目的でない限り、すべてのWebブラウザにJavaScriptエンジンというインタプリタが組み込まれている。今日、HTML、CSSとともにWebを構成する要素の一つである。
HTMLがWebページの基本構造を担当し、CSSがデザインを担当する場合、JavaScriptはクライアントステージでWebページの動作を担当する。
自動車に例えれば、HTMLは自動車の骨格、CSSは自動車の外観、JavaScriptは自動車の動力源であるエンジンと見ることができる。

2. 変数の宣言(Variable declaration)

```
let one = 1;
let a = 'apple';
var two = 2;
var b=  'banana';
const three = 3;
const c = 'carrot'

console.log(one);
console.log(a);
console.log(two);
console.log(b);
console.log(three);
console.log(c);
```

- 結果

```
1
apple
2
banana
3
carrot
```

1) var vs let

```
function vars {
  var x = 1;
  let y = 1;

  if (true) {
    var x = 2; // 上記と同じ「x」であり、上書きされる
    let y = 2; // 上記と違う「y」であり、上書きされない

    console.log('x1: ' + x);
    console.log('y1: ' + y);
  }

  console.log('x2: ' + x);
  console.log('y2: ' + y);
}
```

- 結果

```
x1: 2
y1: 2
x2: 2
x2: 1
```

変数タイプ	var	let	const

スコープ	Function-scoped or global-scoped.	Block-scoped (`{}` の内部).	Block-scoped (`{}` の内部).
再宣言(Re-declaration)	同じスコープ内で可	同じスコープ内で不可	同じスコープ内で不可
再割当(Reassignment)	可	可	不可

2) const

> Const(Constant)は、プログラムの実行中に変更できないデータのコンテナである。

```
const a = 2 * 2;

console.log(a);

a = 3;
```

- 結果

```
4
Uncaught TypeError: Assignment to constant variable.
```

2. スコープ(Scope)

> スコープは変数のアクセス可能性を決定される。
>
> JavaScript 変数には 3 種類のスコープがある。
>
> - Global scope
> - Function scope
> - Block scope

1) Global scope

```
let s = "Squish";

function myFunction() {
  s = "Squish!"
  console.log(s)
}

myFunction()
```

- 結果

```
Squish!
```

2) Function scope

```
function myFunction() {
  let s = "Squish";
}

console.log(s);
```

- 結果

```
console.log(s);
            ^
ReferenceError: s is not defined
```

3) Block Scope

```
if (true) {
  let s = "Squish";
}

console.log(s);
```

- 結果

```
console.log(s);
```

```
            ^
ReferenceError: s is not defined
```

3. データタイプ(Data Type)

1) String

> Stringは文字列、文字のシーケンスである。Javascript の文字列は、一重引用符 (''）、二重引用符 ("")、またはバッククォート (``) (テンプレートリテラル) で囲んで記述される。

```
let s1 = 'Squish.';
let s2 = "Squish?";
let s3 = `Squish!`;

console.log(s1);
console.log(s2);
console.log(s3);
```

- 結果

```
Squish.
Squish?
Squish!
```

2) Number

> NumberはJavaScriptの数値データ型である。

```
let n1 = 37;
let n2 = -9.12;

console.log(n1);
console.log(n2);
```

- 結果

```
37
-9.12
```

3) Boolean

> true または false の 2 つの値のいずれかになる単純なデータ型

```
let x = 5;

console.log(x == 8);
console.log(x != 8);
```

- 結果

```
false
true
```

4) Undefined

> 変数は宣言されているが、初期化されていないか、値が割り当てられていないデータ型

```
let x;

console.log(x);
console.log(typeof(x) === 'string');
console.log(typeof(x) === 'number');
console.log(x === undefined);
```

- 結果

```
undefined
false
false
true
```

5) null

> JavaScript の null 値は、オブジェクト値が意図的に存在しないことを意味する。nullは変数をリセットする方法の一つです。

```
let x = null;

console.log(x);
console.log(x === undefined);
console.log(x === null);
```

- 結果

```
null
false
true
```

6) Object

> JavaScriptのobjectは、キーと値のペア(key-value)を持つデータ構造です。

```
const station19 = {name:"Tachikawa", code:"JC19"};

const station20 = {};

station20.name = "Hino";
station20.code = "JC20";
station20.line = "Chuo";

delete station20.line;

console.log(station19.name);
console.log(station19["code"]);
console.log(station20["name"]);
console.log(station20["line"]);
```

- 結果

```
Tachikawa
JC19
Hino
undefined
```

7) Arrays

> Arrays(配列)は、項目のコレクションを格納し、変数に割り当てることができるオブジェクトです。

```
let fruits = ["Apple", "Orange"];

fruits.push("Plum");
console.log(fruits[0]);
console.log(fruits[2]);

fruits.pop();

console.log(fruits[2]);
```

- 結果

```
Apple
Plum
undefined
```

8) typeof

```
console.log(typeof 42);

console.log(typeof "pineapplepen");

console.log(typeof true);

console.log(typeof undeclaredVariable);

console.log(typeof {name: 'takao'})
```

- 結果

```
number
string
boolean
undefined
object
```

4. ループ(Loops)

1) for

```
let str = "";
let n = 0;

for (let i = 0; i < 9; i++) {
  str += i;
  n++;
}

console.log(str);
console.log(n);
```

- 結果

```
012345678
9
```

2) while

```
let str = "";
let n = 0;

while (n < 9) {
  str += n;
  n++;
}

console.log(str);
console.log(n);
```

- 結果

```
012345678
9
```

3) break & continue

```
let i = 0;

while (i < 9) {
  if (i < 3) {
    console.log("continue")
    i += 1;
    continue;
  } else if (i === 7) {
    break;
  }
  i += 1;
}

console.log(i);
```

- 結果

```
continue
continue
continue
7
```

5. 条件文(IF文)

1) If ... Else

```
function myIf(x) {
  let result;
  if (x > 0) {
    result = "正の数";
  } else if (x < 0) {
    result = "負の数";
  } else {
    result = "ゼロ"
```

```
  }
  return result;
}

let a = -2;
let b = 101.23;
console.log(myIf(a));
console.log(myIf(b));
console.log(myIf(0));
```

- 結果

```
負の数
正の数
ゼロ
```

2) Switch

```
function mySwitch(code) {
  switch (code) {
    case 'JC19':
      console.log('Tachikawa');
      break;
    case 'JC20':
      console.log('Hino');
      break;
    default:
      console.log('etc')
      break;
  }
}

let a = 'JC19';
let b = 'JC01';
let c = '高尾';
mySwitch(a);
mySwitch(b);
mySwitch(c);
```

- 結果

```
Tachikawa
etc
etc
```

6. エラーハンドリング(try-catch文)

1) general

```
function myFunction() {
      return 'kawasaki'
};

try {
    const x = 'yokohama';
    x = myFunction();
} catch (e) {
    console.log(e instanceof TypeError);
    console.log(e instanceof SyntaxError);
    console.log(e.message);
    console.log(e.name);
    console.log(e.stack);
}
```

- 結果

```
true
false
Assignment to constant variable.
TypeError
TypeError: Assignment to constant variable.
```

2) with 'throw'

```
try {
    throw new SyntaxError("Hello");
} catch (e) {
    console.log(e instanceof SyntaxError);
    console.log(e.message);
    console.log(e.name);
    console.log(e.stack);
```

```
}
```

- 結果

```
true
Hello
SyntaxError
SyntaxError: Hello
```

7. 演算子(Operator)

1) 条件演算子 / 三項演算子(Conditional operator)

```
function getFee(isMember) {
  return isMember ? "¥400" : "¥600";
}

console.log(getFee(true));
console.log(getFee(false));
console.log(getFee(null));
```

- 結果

```
¥400
¥600
¥600
```

2) 算術演算子(Arithmetic operator)

Operator	Description
+	足し算(Addition)
-	引き算(Subtraction)
*	掛け算(Multiplication)
**	累乗(Exponentiation)
/	割り算(Division)
%	剰余(Remainder)
++	増加(Increment)
--	減少(Decrement)

```
var x = 0;
var y = 1;
var z = 2;

console.log(x + y);
console.log(x - z);
console.log(z * z);
console.log(z ** 10);
console.log(y / 3);
console.log(z % 5);
console.log(++x); // preincrement; x = 1
console.log(x++); // post increment; x = 2
console.log(x--); // post decrement; x = 1
console.log(--x); // predecrement; x = 0
```

- 結果

```
1
-2
4
1024
0.3333333333333333
2
1
1
2
0
```

3) 比較・等価演算子(Comparison Operator)

a. 比較演算子

演算子	意味
>	大なり
>=	以上
<	少なり
<=	以下

```
const x = 1;
const y = 20;
const z = 100;
const k = 20;

console.log(x > y);
console.log(x > 0);
console.log(y >= k);
console.log(z >= y);
console.log(z <= 199);
console.log('AAA' > 'ABC'); // abc順比較
console.log('earth' > 'mars');  // abc順比較
console.log('あ' < 'け'); // あいうえお順
```

- 結果

```
false
true
true
true
true
false
false
true
```

b. 等価演算子

①値比較; “==” & “!=”

```
let x = 5;
let y = '5';
let z = 10;

console.log(x == z);
console.log(x == y);
```

- 結果

```
false
true
```

②値・タイプ比較; “===” & “!==”

```
let x = 5;
let y = '5';
let z = 10;

console.log(x === z);
console.log(x === y);
```

- 結果

```
false
false
```

4) 論理演算子(Logical Comparator)

演算子	意味
&&	論理積(AND)
||	論理和(OR)

!	論理否定(NOT)

```
const a1 = true && true; // 1
const a2 = true && false; // 2
const a3 = false && true; // 3
const a4 = false && 3 === 4; // 4

console.log(a1 + "\n" + a2 + "\n" + a3 + '\n' + a4);

const o1 = true || true; // 5
const o2 = false || true; // 6
const o3 = true || false; // 7
const o4 = false || 3 === 4; // 8

console.log(o1 + "\n" + o2 + "\n" + o3 + '\n' + o4);

const n1 = !true; // 9
const n2 = !false; // 10
const n3 = !4 === 4; // 11

console.log(n1 + "\n" + n2 + "\n" + n3);
```

- 結果

```
true
false
false
false
true
true
true
false
false
true
false
```

8. 関数(Function)

> 関数定義 (関数宣言や関数定義文 とも呼ばれる。) は function キーワードと、それに続く以下の内容で構成される。。
>
> 関数の名前。
> 関数への引数のリスト。
> 関数を定義する JavaScript の文。

関数の名前: multiply

関数の引数: number1, number2

```
function multiply(number1, number2) {
  return number1 * number2;
}

const x = 4;
const y = 6;
var result = 0;

result = multiply(x, y);

console.log(result);
```

- 結果

```
24
```

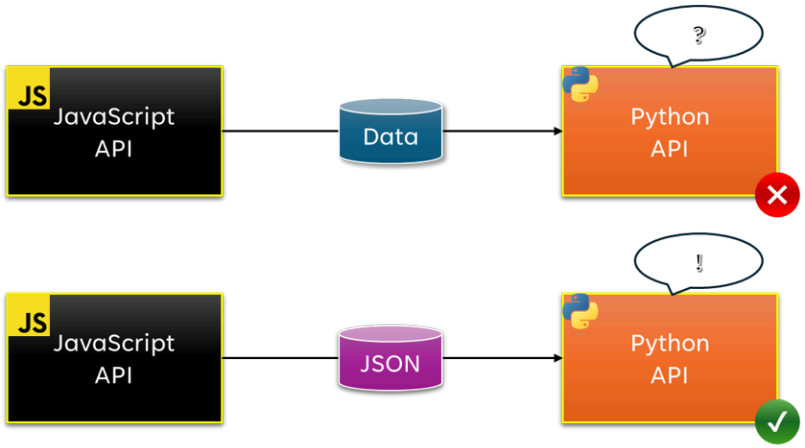
9. 練習問題

🖋 Auto Test Tutorial quiz2.pdf

Ⅲ. プログラミング(Programming)

1. データ交換形式 JSONとXML

- お互い違うシステムのデータ交換のために使う形式である。
- 単純比較ではJSONがXMLより短い、読み書きがやすい。



1) JSON (JavaScript Object Notation)

```
{
  "name": "toyota",
  "number": 21,
  "link": ["hino", "hachioji"]
}
```

2) XML (eXtensible Markup Language)

```
<station>
  <name>toyota</name>
  <number>21</number>
  <link>
    <pre>hino</pre>
    <next>hachioji</next>
  </link>
</station>
```

2. 統合開発環境 (Integrated Development Environment; IDE)

1) IDEの定義

> プログラミングに必要な「ソースコードを書くエディタ」「プログラムを機械語に変換するコンパイラ」「バグを見つけるデバッガ」などのツールを一つにまとめたソフトウェアで、開発作業を効率化し、生産性を高めます。
>
> **IDEの主な機能**
> ソースコードエディタ: 色分け表示（シンタックスハイライト）や自動補完機能でコードを書きやすくする。
> コンパイラ/インタプリタ: コードをコンピュータが実行できる形式に変換する。
> デバッガ: プログラムの実行を一時停止させ、どこに問題（バグ）があるかを見つけやすくする。
> ビルド自動化ツール: コンパイルやテストなどの繰り返し作業を自動化する。
> バージョン管理システムとの連携: Gitなどのバージョン管理システムと連携し、コードの変更履歴を管理しやすくする。

2) デバッグ(Debug)

> ソフトウェアやシステムに潜むエラー（bug; バグ）を見つけ出し、その原因を特定して修正する作業である。
>
> - **バグの発見と特定**: プログラムの動作を観察し、エラーが発生する箇所や条件（再現手順）を突き止める。
> - **原因の解析**: コードの書き間違い（例: totalをtotlaとタイプミス）、ロジックの勘違い、設計ミスなど、バグの根本原因を調査する。
> - **修正**: 特定した原因に基づいてコードを修正します。修正後、新たなバグ（デグレード）が発生していないか確認するテスト（回帰テスト）も重要である。
> - **品質向上**: デバッグは、プログラムの安定性、信頼性、品質を確保するために不可欠なプロセスである。
>
> 詰まり、開発意図と違う動作を修正する行為はすべて「デバッグ」と見なすことができる。

3. 練習問題

Auto Test Tutorial quiz3.pdf