# 2022/06/16 課題 村上航暉

## 課題 1-1

> TypeScript をコンパイルすると、何が生成されるでしょうか？

JavaScript ファイルが生成される。

## 課題 1-2

> JavaScript ファイルの拡張子を「.ts」に変更した後、tsc でコンパイルすると、何が生成されるでしょうか？

JavaScript ファイルが生成される。
(何らかの条件下で、アロー関数ではなく通常関数で記述される？)

## 課題 1-3

> 以下のアンビエント宣言を使用したソースを VS Code で実装して、自動補完が有効になることを確認してください。

```
declare class jQuery {
  html(html: string): void;
}

declare function $(query: string): jQuery;

$('#id').html('Hello World');
```

`declare class` をタイプした段階で、
`declare class name {constructor(parameters) {}}` が補完されることを確認。
また、アンビエント宣言をすることで、ts のエラーが発生しないことを確認。
`declare` が無い場合、 関数の実装がないか、宣言の直後に指定されていません。*ts(2391)* が表示される。

## 課題 1-4

> 上記の最後を以下のように変更した場合の挙動を確認してください。

```
$(123).html('Hello World');
```

型 *'number'* の引数を型 *'string'* のパラメーターに割り当てることはできません。*ts(2345)* が表示される。
`declare function $(query: string): jQuery;` で string 型を指定しているため、TypeScript 側がエラーと認識してくれる。

## 課題 2-1

> 以下の ①、② に当てはまる内容を答えてください。

```
TypeScriptはJavaScriptの①であり、JavaScriptにコンパイラやIDEで使う②が付いただけのものです。
```

①: スーパーセット(上位互換)
②: 型情報

## 課題 2-2

> null と undefined の違いを説明してください。

| **null** | **undefined** |
| --- | --- |
| 代入すべき値が存在しない | 値が代入されていない |
| 意図的に使わない限り発生しない | 自然に発生し得る |
| リテラル | 変数 |
| object | undefined |

参考: undefined と null の違い | TypeScript 入門『サバイバル TypeScript』

## 課題 2-3

> 以下のソースを実際に実行させてください。

```
function createCounter() {
let val = 0;
return {
increment() { val++ },
getVal() { return val }
}
}
let counter = createCounter();
counter.increment();
console.log(counter.getVal()); // 1
counter.increment();
console.log(counter.getVal()); // 2
```

`counter.ts` にて実装し、実行。

## 課題 2-4

> 現在値を取得すると内部の値をインクリメントする関数を実装しました。 コード A、B を実行し、仕様通りに動くのはどちらかを答えてください。 また、結果の違いについて理由を説明してください。

コード A

```
function createSequence() {
  let val = 1;
  return function () {
    return val++;
  };
}

const sequence = createSequence();
console.log(sequence());
console.log(sequence());
```

コード B

```
function createSequence() {
  let val = 1;
  return val++;
}

const sequence = createSequence;
console.log(sequence());
console.log(sequence());
```

仕様通りに動作するのは **コード A**。
コード A は `sequence()` で `return function()` を呼び出しているが、
コード B が `sequence()` で呼び出しているのは `createSequence()` 自体であるため、呼び出しの度に変数 `val` が初期化されている。

## 課題 3-1

> 以下の正方形クラスの実装を完成させてください。

```
/**
 * 四角形のインターフェイス。
 */
interface Quadrilateral {
  /**
   * @returns 面積
   */
  findArea(): number;
}

/**
 * 点
 */
class Point {
  constructor(public x: number, public y: number) {}
}
```

```
/**
 * 正方形。
 */
class Square implements Quadrilateral {
  /**
   * コンストラクタ。
   *
   * @param p1 左上の点
   * @param p2 右下の点
   */
  constructor(private p1: Point, private p2: Point) {}

  /**
   * @returns 1辺の長さ
   */
  private getSideLength(): number {
    // 対角線の長さ = p1とp2の距離
    const diagonal = Math.sqrt(
      Math.pow(this.p2.x - this.p1.x, 2) + Math.pow(this.p2.y - this.p1.y, 2)
    );
    // 正方形なので、対角線の平方根を返す。
    return diagonal / Math.sqrt(2);
  }

}

const square = new Square(new Point(0, 2), new Point(2, 0));
console.log(square.findArea());
```

square.ts にて実装。

## 課題 3-2

> https://typescript-jp.gitbook.io/deep-dive/future-javascript/arrow-functions の例を元に、通常の function とアロー関数での this の挙動の違いを説明してください。

> 通常の functon では、this は呼び出しコンテキストになります。

| 違い | 通常関数 | アロー関数 |
| --- | --- | --- |
| コンテキスト | 呼び出しコンテキスト | アロー関数を囲んだコンテキスト |
| 内部の this | 実行時のレシーバであるオブジェクト | 宣言時のスコープを持つオブジェクト |

参考: アロー関数と関数の違いと this の評価のされ方 - Qiita

## 課題 3-3

> https://typescript-jp.gitbook.io/deep-dive/future-javascript/let を元に、var と、let および const のスコープを説明してください。

| var | let / const |
| --- | --- |

| var | let / const |
| --- | --- |
| 関数スコープ | ブロックスコープ |

`var` は関数スコープのため、 `{}` の中と外の同一名変数は同じものを指している。
関数スコープなので、 `function hoge() {}` の中と外の同一名変数は別のものを指す。
`let` と `const` はブロックスコープのため、 `{}` の中と外は異なる変数となる。

## 課題 3-4

> https://typescript-jp.gitbook.io/deep-dive/future-javascript/for...of の例を元に、for ... in と for ... of の挙動を確認してください。

`forInForOf.ts` にて挙動を確認。

## 課題 3-5

> 以下のソースのジェネレータを、i が 100 になったら終了するように修正してください。

```
function* fizzbuzzGenerator() {
  let i = 0;
  while (true) {
    i++;
    if (i % 15 === 0) {
      yield 'FizzBuzz';
      continue;
    }
    if (i % 3 === 0) {
      yield 'Fizz';
      continue;
    }
    if (i % 5 === 0) {
      yield 'Buzz';
      continue;
    }
    yield '' + i;
  }
}

const iterator = fizzbuzzGenerator();
while (true) {
  const e = iterator.next();
  if (e.done) {
    break;
  }
  console.log(e.value);
}
```

`fizzbuzzGenerator.ts` にて修正を実装。
参考: イテレーターとジェネレーター - JavaScript | MDN