Christian

1. PE01.HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <meta name="author" content="Chris M." />

    <link rel="stylesheet" type="text/css" href="PE01.css" />

    <title>Programming Exercise #1</title>
  </head>

  <a href="https://www.cityu.edu/">Visit CityU.edu Now!</a>

  <body>
    <h2>“Hello City University of Seattle!”</h2>
    <marquee> “Visit CityU Today!” </marquee>
    <marquee
      direction="left"
      behavior="alternate"
      style="border:blue 1px SOLID"
    >
    </marquee>
  </body>
</html>
```

2. PE01.css

```
a:link {
  color: blue;
}

/* visited link */
a:visited {
  color: purple;
}

/* mouse over link */
a:hover {
  color: hotpink;
}

/* selected link */
a:active {
```

```
  color: green;
}
```

PE03

1. PE03.ts

```
let firstName;
let lastName;
let fullName;
let age;
let ukCitizen;

firstName = "Rebecca";
lastName = "Smith";
age = 42;
ukCitizen = false;
fullName = firstName + " " + lastName;

if (ukCitizen) {
  console.log(
    "My name is " +
      fullName +
      ", I'm " +
      age +
      ", and I'm a citizen of the United Kingdom."
  );
} else {
  console.log(
    "My name is " +
      fullName +
      ", I'm " +
      age +
      ", and I'm not a citizen of the United Kingdom."
  );
}
let x;
let y;
let a;

x = "five";
y = 7;
a = x + y;

console.log(a);

let randomNumbers = [1, 10];
let nextNumber = x;
for (let i = 0; i < 10; i++) {
  nextNumber = Math.floor(Math.random() * (100 - 1)) + 1;
  randomNumbers[i] = nextNumber;
```

```
}
console.log(randomNumbers);
```

2. Some.ts

```
function greaterThan(n: number) {
  return (m) => m > n;
}
console.log(greaterThan(11)(10));
```

PE05

1. inputValidation.ts

```
function Input() {
  let input = console.log("input a number between 1 and 20.");
  return validateInput(input);
}
function validateInput(input) {
  if (input >= 1 && input <= 20 && !isNaN(input)) {
    return input;
  } else {
    console.log("Wrong number!");
    return Input();
  }
}
let input = Input();
console.log(input);
function someFunction(parameter1, parameter2) {
  if (parameter1 <= 0) {
    return parameter2;
  } else {
    parameter2 *= parameter1;
    parameter1--;
    return someFunction(parameter1, parameter2);
  }
}
let output = someFunction(5, 2);
console.log(output);
const a = parseInt("Input Value: ");
let b = 0,
  c = 1,
  d;
console.log("Fibonacci Sequence:");
for (let i = 1; i <= a; i++) {
  console.log(b);
  d = b + c;
  b = c;
  c = d;
```

```
}
console.log("Program end");
```

PE06

1. PE06.ts

```
interface Pizza {
  type: string;
  slices: number;
  crust: string;
}
const myPizza: Toppings = {
  type: "Capsicum",
  slices: 7,
  sauce: "alfredo",
};
function compareValue(small: number, large: number): number {
  const rem = large % small;

  if (rem == 0) {
    return small;
  }
  return compareValue(rem, small);
}
function checkSlices(pizza: Pizza) {
  if (pizza.slices > 8) {
    return "Number is too high";
  }
  const Value = compareValue(pizza.slices, 8);
  const portion =
    pizza.slices == 8 ? "1" : `${pizza.slices / Value}/${8 / Value}`;
  return `There is ${portion} remaining of the pizza`;
}
console.log(checkSlices(myPizza));
interface Toppings extends Pizza {
  sauce: "tomato" | "alfrado" | "bbq sauce";
  pineapple?: boolean;
  parmesan?: boolean;
  crust?: string;
}
```

PE07

1. DefineClass.ts

```
class processIdentity<T, U> {
  private value: T;
  private message: U;
  constructor(value: T, message: U) {
```

```
    this.value = value;
    this.message = message;
  }
  getIdentity(): T {
    console.log(this.message);
    return this.value;
  }
}

var processor = new processIdentity(5, "Hello World");
console.log(processor.getIdentity());
```

2. GenericClassType.ts

```
interface ProcessIdentity<T, U> {
  value: T;
  message: U;
  process(): T;
}

class processIdentity<X, Y> implements ProcessIdentity<X, Y> {
  value: X;
  message: Y;
  constructor(value: X, message: Y) {
    this.value = value;
    this.message = message;
  }

  process(): X {
    return this.value;
  }
}

var processor = new processIdentity(5, "Hello World");
console.log(processor.process());
processor.value = 5;
console.log(processor.process());
```

3. GenericType.ts

```
interface Identity<T, U> {
  value: T;
  message: U;
}

var object1: Identity<number, string> = {
  value: 5,
  message: "Hello World",
};
var object2: Identity<string, number> = {
```

```
  value: "Hello World",
  message: 5,
};

interface ProcessIdentity<T, U> {
  (value: T, message: U): T;
}

function processIdentity<T, U>(value: T, message: U): T {
  console.log(message);
  return value;
}

var processor1: ProcessIdentity<number, string> = processIdentity;
var processor2: ProcessIdentity<string, number> = processIdentity;

var returnNumber1 = processor1(5, "Hello");
var returnNumber2 = processor2("Hello", 5);
```

PE08

```

```

```

```

PE09

```

```

```

```

PE10