## Учреждение образования

## Белорусский государственный технологический университет

Кафедра полиграфического оборудования и

системы обработки информации

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

Тема

«Ознакомление со стандартами языков программирования

в части спецификаций *ECMA* 5 и *ECMA* 6»

по дисциплине «Стандартизация и сертификация

информационных систем и технологий»

Выполнил студент

ФИТ 4 курс 7 группа Бобрович Г.С.

Проверил

кандидат технических наук

Сулим П.Е.

Отчет по лабораторной работе

защищен с отметкой баллов

Минск 2024

***Цель* *работы*:**

Написать код реализующий необходимый функционал на *ES*5, используя *Node.js* или браузер *Google Chrome.*

***Ход работы:***

Необходимо ознакомится с *ES*6 и *ES*5, привести основные различия. Переписать аналогичный функционал на *ES*5, используя *Node.js* или браузер *Google Chrome*.

[ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript) — встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков.

[ECMAScript 5](https://es5.javascript.ru) (ES5):

1. Поддержка строгого режима (strict mode). Это специальный режим, который делает код более строгим и помогает избегать ошибок.
2. Аксессоры getters и setters. Это способы получения и установки значений объектов.
3. Возможность использовать зарезервированные слова в качестве ключей свойств и ставить запятые в конце массива.
4. Многострочные строковые литералы.
5. Новая функциональность в стандартной библиотеке.
6. Поддержка JSON. JSON — это формат данных, который часто используется для передачи данных между клиентом и сервером.

[ECMAScript 6](https://www.w3schools.com/js/js_es6.asp) (ES6):

1. Деструктурирующее присваивание. Это присваивание значения нескольким переменным одновременно.
2. Стрелочные функции. Это новый, более короткий способ написания функций.
3. Объявление строк с помощью обратных кавычек (`). Шаблонные строки могут быть многострочными, также могут интерполироваться. Это новый способ создания строк, который позволяет включать переменные прямо в строку.
4. let и const — альтернативы var для объявления переменных. Добавлена «временная мертвая зона». Это новые способы объявления переменных, которые предоставляют больше контроля над тем, как и когда вы можете изменять эти переменные.
5. Итератор и протокол итерации теперь определяют способ перебора любого объекта, а не только массивов. Symbol используется для присвоения итератора к любому объекту.
6. Функции-генераторы. Это специальный тип функций, который может приостанавливать свое выполнение и возобновлять его позже. Они используют yield для создания последовательности элементов. Функции-генераторы могут использовать yield\* для делегирования в другую функцию генератора, кроме этого, они могут возвращать объект генератора, который реализует оба протокола.

Ниже показаны различия кода на разных версиях ECMA. Красным цветом выделены изменения.

|  |  |
| --- | --- |
| ECMA 6 | ECMA 5 |
| const NPC = require("./NPC");  const MeleeEnemy = require("./MeleeEnemy");  const settings = require("./settings");  let wolf1 = new MeleeEnemy(  settings.HardWolf.name,  settings.HardWolf.lvl,  );  let wolf2 = new MeleeEnemy(  settings.SimpleWolf.name,  settings.SimpleWolf.lvl,  settings.SimpleWolf.attackStrength,  settings.SimpleWolf.attackSpeed  );  let NPCExample = NPC.create();  console.log( wolf1 );  console.log( wolf2 );  console.log( NPCExample );  console.log( wolf1.name );  console.log( ); | var NPC = require("./NPC");  var MeleeEnemy = require("./MeleeEnemy");  var settings = require("./settings");  var wolf1 = new MeleeEnemy(  settings.HardWolf.name,  settings.HardWolf.lvl,  );  var wolf2 = new MeleeEnemy(  settings.SimpleWolf.name,  settings.SimpleWolf.lvl,  settings.SimpleWolf.attackStrength,  settings.SimpleWolf.attackSpeed  );  var NPCExample = NPC.create();  console.log( wolf1 );  console.log( wolf2 );  console.log( NPCExample );  console.log( wolf1.name );  console.log( ); |
| class NPC {  constructor(name = "NPC", pos = 0) {  this.\_name = name;  this.\_pos = pos;  NPC.count++;  }  get name() {  return this.\_name;  }  set name(value) {  this.\_name = value;  }  get pos() {  return this.\_pos;  }  static create() {  return new NPC();  }  walkForward() {  this.\_pos++;  }  walkBack() {  this.\_pos++;  }  }  NPC.count = 0;  if (module.exports) {  module.exports = NPC;  } else {  throw new Error("Use Node.js!");  } | function NPC(name, pos) {  this.\_name = name || "NPC";  this.\_pos = pos || 0;  NPC.count++;  }  NPC.prototype.name = function(value) {  if (value === undefined) {  return this.\_name;  } else {  this.\_name = value;  }  };  NPC.prototype.pos = function() {  return this.\_pos;  };  NPC.create = function() {  return new NPC();  };  NPC.prototype.walkForward = function() {  this.\_pos++;  };  NPC.prototype.walkBack = function() {  this.\_pos--;  };  NPC.count = 0;  if (typeof module !== 'undefined' && module.exports) {  module.exports = NPC;  } else {  throw new Error("Use Node.js!");  } |
| const NPC = require("./NPC.js");  class MeleeEnemy extends NPC {  constructor(name, pos, lvl, attackStrength, attackSpeed) {  super(name, pos);  this.lvl = lvl;  this.attackStrength = attackStrength;  this.attackSpeed = attackSpeed;  }  walkForward() {  super.walkForward();  console.log('--> Argh!');  }  walkBack() {  super.walkBack();  console.log('<--')  }  }  if (module.exports) {  module.exports = MeleeEnemy;  } else {  throw new Exception("Use node.js!");  } | var NPC = require("./NPC.js");  function MeleeEnemy(name, pos, lvl, attackStrength, attackSpeed) {  NPC.call(this, name, pos);  this.lvl = lvl;  this.attackStrength = attackStrength;  this.attackSpeed = attackSpeed;  }  MeleeEnemy.prototype = Object.create(NPC.prototype);  MeleeEnemy.prototype.constructor = MeleeEnemy;  MeleeEnemy.prototype.walkForward = function() {  NPC.prototype.walkForward.call(this);  console.log('--> Argh!');  };  MeleeEnemy.prototype.walkBack = function() {  NPC.prototype.walkBack.call(this);  console.log('<--');  };  if (typeof module !== 'undefined' && module.exports) {  module.exports = MeleeEnemy;  } else {  throw new Error("Use node.js!");  } |
| {  "SimpleWolf": {  "name": "SimpleWolf",  "lvl": 2,  "attackStrength": 2,  "attackSpeed": 4  },  "HardWolf": {  "name": "HardWolf",  "lvl": 4,  "attackStrength": 4,  "attackSpeed": 8  }  } | {  "SimpleWolf": {  "name": "SimpleWolf",  "lvl": 2,  "attackStrength": 2,  "attackSpeed": 4  },  "HardWolf": {  "name": "HardWolf",  "lvl": 4,  "attackStrength": 4,  "attackSpeed": 8  }  } |

Приведите выводы по содержанию работы.

Вывод: были рассмотрены две версии спецификации ECMA, выявлены их отличия и был переписан предоставленный код.