プロ検
プログラミング能力検定

2022年6月6日改定

プログラミング能力の共通評価指標

プロ検レベル	区分	概念	概念の説明	ビジュアルブロック	テキストコード	JavaScriptでの記述例

レベル1	制御	順次実行	プログラムは上から順番に実行されることを理解している	スタートをクリックしたとき		console.log("1");
						console.log("2");
						console.log("3");

レベル1	制御	条件分岐(if文)	if文を使用して条件によって動作を変えることができる	もし〇〇なら	if ()	let num = 1;
						if (num == 1) {
						  console.log("Hello World!");
						}

レベル1	制御	条件分岐(if~else文)	if～else文を使用して条件に合っている場合と合っていない場合で動作を分けることができる	もし〇〇なら～でなければ	if () {
					} else {
					}
						let num = 1;
						if (num == 0) {
						  console.log("Hello World!");
						} else {
						  console.log("if文の条件にあてはまりません");
						}

レベル1	制御	条件分岐(if~else if文)	if～else if文を使用して複数の条件によって動作を変えることができる	もし〇〇なら,
				もし〇〇なら～でなければ
					if () {
					} else if () {
					}
						let num = 1;
						if (num == 0) {
						  console.log("numは0です");
						} else if (num == 1) {
						  console.log("numは1です");
						}

レベル1	制御	データ出力	値を出力することができる	〇〇という	console.log	console.log("1");

レベル1	数的処理	正負の数	正負の数を扱うことができる	［-10］歩動かす		let num = 10;
						num = num + 5;
						num = num + (-5);

レベル1	数的処理	数字の大小	数字の大小で動作が変わることを理解している	大きさを［20］にする,
				大きさを［100］にする
						let num = 10;
						num = num + 5;
						num = num + 50;

レベル1	演算子	四則演算	四則演算を行うことができる	〇 + 〇
				〇 - 〇
				〇 × 〇
				〇 / 〇
					+, -, *, /	let num = 0;
						num = 2 + 1;
						num = 2 - 1;
						num = 2 * 1;
						num = 2 / 1;

レベル1	演算子	等号	値や文字が「等しいとき」という条件を作成することができる	〇＝〇	==	let str = "dog";
						if (str == "dog") {
						  console.log("いぬ");
						}

レベル1	演算子	不等号	値が「～より大きい（小さい）とき」という条件を作成することができる	〇<〇
				〇>〇
					<, >	let num = 6;
						if (num > 5) {
						  num = 10;
						} else {
						  num = 0;
						}

レベル1	変数	変数の宣言	変数を宣言することができる	変数を作る	let	let num;
レベル1	変数	変数代入	変数に値を代入することができる	変数を〇にする	=	let num = 1; num = 2;

レベル2	制御	反復処理(for文)	for文を使用して処理を反復実行することができる	ずっと,〇回くりかえす	for ()	for (let i = 0; i < 10; i++) {
						  console.log(i);
						}

レベル2	制御	データ入力	ユーザが入力した値を取得することができる	あなたのお名前は？と聞いて待つ		
レベル2	演算子	文字列連結	文字列を連結することができる	〇と〇	" " + " "	let str = "basket" + "ball";

レベル2	演算子	論理積	「複数の条件が全て満たされたとき」という条件を作成することができる	〇〇かつ〇〇	&&	num = 2;
						if (num + 2 == 4 && num * 2 == 4) {
						  console.log("OK");
						}

レベル2	演算子	論理和	「複数の条件のうちいずれかが満たされたとき」という条件を作成することができる	〇〇または〇〇	||	num = 3;
						if (num + 2 == 6 || num * 2 == 6) {
						  console.log("OK");
						}

レベル2	演算子	否定	「条件を満たさなかったとき」という条件を作成することができる	〇〇でない	!	num = 2;
						if (! (num + 2 == 0)) {
						  console.log("OK");
						}

レベル2	演算子	不等価	値や文字が「等しくないとき」という条件を作成することができる	〇〇でない	!=	n = 2;
						if (num * 2 != 0) {
						  console.log("OK");
						}

レベル2	配列	配列宣言	配列を宣言することができる	リストを作る	let arr = [];	let fruits = [];
レベル2	配列	配列の代入	配列に別の配列を代入することができる		let arr = [1, 2]; arr= [0, 1];	let fruits = []; fruits = ["りんご", "バナナ"];
レベル2	配列	値の取得	添字を指定することで、要素の値を取得することができる	リストの〇番目	arr[0]	let fruits = ["りんご", "バナナ", "ぶどう"]; console.log(fruits[0]);
レベル2	配列	配列の長さ	配列の長さを取得することができる	リストの長さ	length	let fruits = ["りんご", "バナナ", "ぶどう"]; console.log(fruits.length);

レベル3	変数	スコープ	定義された変数や関数を利用できる有効範囲を理解している	グローバル・ローカル変数		sayNumber();
						function sayNumber() {
						  let num = 10;
						  console.log(num); //出力される
						}
						console.log(num); //出力されない

レベル3	変数	変数型(数値,文字列)	変数に型があることを理解している			

レベル3	変数	変数型(boolean)	true,falseを使用して、if文などの評価を行うことができる		let bool = true;
					if (bool) {
					}
						let bool = true;
						if (bool) {
						 console.log(bool);
						}

レベル3	変数	キャスト	変数の型を変更することができる		parseInt Number String	str = "10"; let num = parseInt(str);

レベル3	関数	関数	関数を宣言し、呼び出すことができる	関数を作る	function func() {
					}
					func();
						function sayHello() {
						  console.log("hello");
						}
						sayHello();

レベル3	関数	引数	呼び出し元から関数に値を渡すことができる	数字・文字の引数,
				条件の引数
					function func(num) {
					}
					func(num);
						function sayNumber(num) {
						  console.log(num);
						}
						sayNumber(10);

レベル3	関数	返り値	関数から呼び出し元に値を返すことができる		function func(num) {
					  return num + 1;
					}
						function add(num1, num2) {
						  return num1 + num2;
						}
						add(1, 2);

レベル3	配列	反復処理	配列の要素を順番に取り出すことができる	リストの"変数"番目	for (.. of)
					forEach
						let fruits = ["りんご", "バナナ", "ぶどう"];
						for (let fruit of fruits) {
						  console.log(fruit);
						}

レベル3	数的処理	乱数	乱数を作成し、使用することができる	〇から〇までのらんすう	Math.random	let num = Math.floor(Math.random() * 5);
						console.log(num);
						num = Math.floor(Math.random() * 5);
						console.log(num);

レベル4	制御	反復処理(while文)	while文を使用して処理を反復実行することができる	○までくりかえす	while ()
					do{
					}while ();
						while (let i < 10) {
						  console.log(i);
						}

レベル4	制御	反復スキップ処理	現在の反復処理の文の実行を終了し、次の反復処理を実行することができる		continue	for (let i = 0; i < 20; i++) {
						  if (i != 10) {
						  continue;
						  }
						  console.log("10です");
						}

レベル4	制御	反復抜け出し処理	反復処理やswitch文を中断して、次の処理を実行することができる		break	for (let i = 0; i < 1500; i++) {
						  if (i > 1000) {
						  console.log("1000より大きくなりました");
						  break;
						  }
						  console.log(i);
						}

レベル4	制御	条件分岐(switch文,case文)	switch文、case文を使用して、条件によって動作を変えることができる		switch()
					case
						switch (num) {
						  case 1:
						  console.log("numは1です");
						  break;
						  case 2:
						  console.log("numは2です");
						  break;
						  case 3:
						  console.log("numは3です");
						  break;
						}

レベル4	変数	定数宣言	定数を宣言、使用することができる		const	const NUM = 10;

レベル4	配列	要素の削除	配列の値を消去することができる	リストの○番目をけす	splice
					shift
					pop
						let fruits = ["りんご", "バナナ", "ぶどう"];
						fruits.splice(1, 1);
						fruits.shift();
						fruits.pop();

レベル4	配列	要素の追加	配列に値を追加することができる	リストに○○をついかする リストのさいご番目に○○をいれる	splice push	let fruits = ["りんご", "バナナ"]; fruits.push("ぶどう");
レベル4	配列	要素の探索	配列に指定した値が含まれるかどうか確認することができる	リストに○○がふくまれる リスト中の○○の場所	includes indexOf	let fruits = ["りんご", "バナナ", "ぶどう"]; console.log(fruits.includes("りんご")); console.log(fruits.indexOf("りんご"));

レベル5	制御	例外処理	例外が発生したときの処理を指定することができる		try...catch,
					throw,
					error
						try {
						  console.log(n);
						} catch(error) {
						  console.log("error");
						  console.log(error.message);
						}

レベル5	数的処理	剰余	除算の余りを求めることができる	○を○でわったあまり	%	let num = 0; num = 10 % 3;
レベル5	変数	null	値が存在しないことを理解している		null	let value = null;
レベル5	変数	型を調べる	指定した値や変数の型を調べることができる		typeof	let value = 1; console.log(typeof value);

レベル5	変数	型の比較	「値と型が等しいとき」という条件を作成することができる		===	let value = 1;
						if (value === 1) {
						  console.log("int");
						} else if (value === "1") {
						  console.log("string");
						}

レベル5	配列	要素の挿入	配列の指定の要素に値を挿入することができる	リストの○番目に○○をいれる	splice	let fruits = ["りんご", "バナナ", "ぶどう"]; fruits.splice(2, 0, "みかん");
レベル5	配列	要素の代入	配列の指定の要素に値を代入することができる	リストの○番目を○○でおきかえる	splice arr[0] = num;	let fruits = ["りんご", "バナナ", "ぶどう"]; fruits.splice(1, 1, "みかん"); fruits[2] = "もも";

レベル5	配列	多次元配列	配列の要素の中に配列を格納することができる		[
					[ ],
					[ ]
					];
						let fruits = [
						  ["りんご", "アメリカ", 1],
						  ["バナナ", "インド", 2]
						];

レベル5	配列	順序の反転	配列の要素の並び順を反転させることができる		reverse	let number = [1, 5, 2, 3, 4]; number.reverse();
レベル5	配列	配列のソート	ソート機能を用いて配列の要素を並べ替えることができる		sort	let number = [1, 5, 2, 3, 4]; number.sort();

レベル6	制御	並列処理・非同期処理	複数の動作を同時に実行することができる	スクリプトを2つ使用	setTimeout,
					setInterval,
					clearInterval
						console.log("1");
						setTimeout(two, 1000);
						console.log("3");
						
						function two() {
						  console.log("2");
						}

レベル6	オブジェクト指向	連想配列の生成	連想配列を生成することができる		let arr = {
					  key: value;
					}
						let fruit = {
						  name: "りんご",
						  area: "アメリカ",
						  production: 500
						}

レベル6	オブジェクト指向	連想配列の値の取得	連想配列の指定したキーの値を参照することができる		arr.key	let fruit = {
						  name: "りんご",
						  area: "アメリカ",
						  production: 500
						}
						console.log(fruit.name);

レベル6	オブジェクト指向	連想配列の反復処理	連想配列の要素を繰り返し取得することができる		for..in	let fruit = {
						  name: "りんご",
						  area: "アメリカ",
						  production: 500
						}
						for (let key in fruit) {
						  if (key == "name") {
						  console.log(fruit[key]);
						  }
						}

レベル6	オブジェクト指向	クラスの定義	クラスを定義し、インスタンスを生成することができる		class	class Food {
						  sayApple() {
						  console.log("りんご");
						  }
						}
						food = new Food;
						food.sayApple();

レベル6	オブジェクト指向	コンストラクタ	コンストラクタを使用して、インスタンス生成時にメソッドを自動的に実行することができる		constructor	class Food {
						  constructor(name, area) {
						  this.name = name;
						  this.area = area;
						  }
						}
						let food = new Food("りんご", "アメリカ");

レベル6	オブジェクト指向	クラス内の変数	クラス内で宣言する変数を活用することができる		this.num	class Food {
						  constructor(name, area) {
						  this.name = name;
						  this.area = area;
						  }
						}
						let food = new Food("りんご", "アメリカ");
						console.log(food.name);

レベル6	オブジェクト指向	メソッド	クラスのメソッドを活用することができる		class Sample{
					　method() {
					　}
					}
					let sample = new Sample;
					sample.method();
						class Food {
						　constructor(name, area) {
						　　this.name = name;
						　　this.area = area;
						　　this.price = 0;
						　}
						　setPrice(price) {
						　　this.price = price;
						　}
						　getPrice() {
						　　return this.price;
						　}
						}
						let food = new Food("りんご", "アメリカ");
						food.setPrice(100);
						console.log(food.getPrice());

レベル6	オブジェクト指向	クラスの継承	クラスを継承し、クラス内の変数やメソッドを活用することができる		extends	class Food {
						　constructor(name, area) {
						　　this.name = name;
						　　this.area = area;
						　　this.price = 0;
						　}
						　setPrice(price) {
						　　this.price = price;
						　}
						　getPrice() {
						　　return this.price;
						　}
						}
						
						class Fruit extends Food {
						　getTaste() {
						　　return "甘い";
						　}
						}
						let fruit = new Fruit("りんご", "アメリカ");
						fruit.setPrice(100);
						console.log(fruit.getPrice());
						console.log(fruit.getTaste());

レベル6	オブジェクト指向	オーバーライド	クラスを継承し、メソッドを上書きすることができる		method(){
					　super.method();
					}
						class Food {
						　constructor(name, area) {
						　　this.name = name;
						　　this.area = area;
						　　this.price = 0;
						　}
						　setPrice(price) {
						　　this.price = price;
						　}
						　getPrice() {
						　　return this.price;
						　}
						}
						
						class Fruit extends Food {
						　constructor(name, area, taste) {
						　　super(name, area);
						　　this.taste = taste;
						　}
						}
						let fruit = new Fruit("りんご", "アメリカ", "甘い");
						fruit.setPrice(100);
						console.log(fruit.taste);