Editor javascript:

```
// Change code below this line

const productName = "Droid";
const pricePerItem = 2000;

// console.log(productName);
// 'Droid'

// console.log(pricePerItem);
// 2000

```

Theory | **Task and tests** | Result — Done — Assignment 1/36

Declare two variables, `productName` for the product name and `pricePerItem` for storing the price per item. When declaring, assign the following product characteristics to the variables:

- name - string `"Droid"`
- price per item - number `2000`

TESTS

- The variable `productName` is declared
- The value of the variable `productName` is the string `"Droid"`
- The variable `pricePerItem` is declared
- The value of the variable `pricePerItem` is the number `2000`

Editor javascript:

```
let productName = "Droid";
let pricePerItem = 2000;

// Change code below this line
productName = "Repair droid";
pricePerItem = 3500;
```

Theory | **Task and tests** | Result — Done — Assignment 2/36

The name of the item has been changed to `"Repair droid"` and its price was increased by `1500` credits. Override the values of the `pricePerItem` and `productName` variables after they are declared.

TESTS

- The variable `pricePerItem` is declared with `let`
- When declaring, the value - the number 2000 - is assigned to the variable `pricePerItem`
- The new value that is 1500 more than the previous one is assigned to the variable `pricePerItem`
- The variable `productName` is declared with `let`
- When declaring, the value - the string `"Droid"` - is assigned to the variable `productName`
- The new value - the string `"Repair droid"` - is assigned to the variable `productName`

Editor javascript:

```
// Change code below this line
let topSpeed = 160;
let distance = 617.54;
let login = "mango935";
let isOnline = true;
let isAdmin = false;
```

Theory | **Task and tests** | Result — Done — Assignment 3/36

Declare the following variables using the `const` or `let` keyword and assign the appropriate values to them.

- `topSpeed` - number `160`.
- `distance` - number `617.54`.
- `login` - string `"mango935"`.
- `isOnline` - boolean `true`.
- `isAdmin` - boolean `false`.

TESTS

- The variable `topSpeed` is declared
- The value of the `topSpeed` variable is `160`
- The variable `distance` is declared
- The value of the `distance` variable is the number `617.54`

Editor javascript:

```
const pricePerItem = 3500;
const orderedQuantity = 4;

// Change code below this line
const totalPrice = pricePerItem * orderedQuantity;

```

Theory | **Task and tests** | Result — Done ← Assignment 4/36 →

Complete the code by assigning to the variable `totalPrice` an expression for calculating the total amount of the order. The variable `pricePerItem` stores the price of one product unit, and `orderedQuantity` - the number of product units in the order.

TESTS

- The variable `pricePerItem` is declared
- The value of the variable `pricePerItem` is the number `3500`
- The variable `orderedQuantity` is declared
- The value of the variable `orderedQuantity` is the number `4`
- The variable `totalPrice` is declared
- The value of the variable `totalPrice` is the number `14000`
- Operator `*` used

Editor javascript:

```
const productName = "Droid";
const pricePerItem = 3500;

// Change code below this line
const message = `You picked ${productName}, price per item is ${pricePerItem} credits`;
```

Theory | **Task and tests** | Result — Done ← Assignment 5/36 →

Declare the variable `message` and write in it a message about the purchase, a string in the format: `"You picked <product name>, price per item is <product price> credits"`. Where `<product name>` and `<product price>` are the values of the `productName` and `pricePerItem` variables. Use template string syntax.

TESTS

- The variable `productName` is declared
- The value of the variable `productName` is the string `"Droid"`
- The variable `pricePerItem` is declared
- The value of the variable `price` is the number `3500`
- The variable `message` is declared
- The value of the variable `message` is the string `"You picked Droid, price per item is 3500 credits"`

Editor javascript:

```
// Change code below this line
const pricePerDroid = 800;
let orderedQuantity = 6;
const deliveryFee = 50;
const totalPrice = pricePerDroid * orderedQuantity + deliveryFee;
const message = `You ordered droids worth ${totalPrice} credits. Delivery (${deliveryFee} credits) is included in total price.`

```

Theory | **Task and tests** | Result — Done ← Assignment 6/36 →

A shop selling repair droids is ready to open, it remains to write a script to order them. Declare the variables and assign the appropriate values to them:

- `pricePerDroid` - price per one droid, value `800`
- `orderedQuantity` - a number of droids in the order, value `6`
- `deliveryFee` - delivery fee, value `50`
- `totalPrice` - total price, do not forget about the delivery fee
- `message` - message about the status of the order in the format `"You ordered droids worth <total price> credits. Delivery (<delivery fee> credits) is included in total price."`

TESTS

- The variable `orderedQuantity` is declared
- The value of the `orderedQuantity` variable is the number `6`
- The variable `pricePerDroid` is declared

Editor javascript:

```
// Change code below this line
function sayHi() {
  console.log("Hello, this is my first function!");
}
sayHi();
```

Theory | **Task and tests** | Result — Done — Assignment 7/36

Declare a `sayHi` function, inside which you should add `console.log()` with the string `"Hello, this is my first function!"`. After the declaration, call the `sayHi` function.

TESTS

- Function declaration expected
- The name `sayHi` is assigned to the function
- The body of the `sayHi` function contains `console.log("Hello, this is my first function!")`
- After the declaration, there is a call to the `sayHi` function

Editor javascript:

```
// Change code below this line
function add(a, b, c) {
  console.log(`Addition result equals ${a + b + c}`);
  // Change code above this line
}

add(15, 27, 10);
add(10, 20, 30);
add(5, 10, 15);
```

Theory | **Task and tests** | Result — Done — Assignment 8/36

The `add` function must be able to add three numbers and output the result to the console. Add three parameters to the function `add`, `a`, `b` and `c`, which will receive the values of the arguments when it is called.

Complete `console.log()` to log the string `"Addition result equals <result>"`, where `<result>` is the sum of the passed numbers.

TESTS

- The function `add(a, b, c)` is declared
- The call `add(15, 27, 10)` outputs `"Addition result equals 52" to the console`
- The call `add(10, 20, 30)` outputs `"Addition result equals 60" to the console`
- The call `add(5, 10, 15)` outputs `"Addition result equals 30" to the console`

Editor javascript:

```
function add(a, b, c) {
  // Change code below this line

return a + b + c;


  // Change code above this line
}

add(2, 5, 8); // 15

console.log(add(15, 27, 10));
console.log(add(10, 20, 30));
console.log(add(5, 10, 15));
```

Theory | **Task and tests** | Result — Done — Assignment 9/36

Modify the code of the `add` function so that it returns the result of adding the values of the three parameters `a`, `b` and `c`.

TESTS

- The function `add(a, b, c)` is declared
- The `add` function has a `return` operator
- The call `add(15, 27, 10)` returns `52`
- The call `add(10, 20, 30)` returns `60`
- The call `add(5, 10, 15)` returns `30`
- Calling a function with random but valid arguments returns the correct value

Editor javascript:

```
function makeMessage (name, price) {
  // Change code below this line
   const message = `You picked ${name}, price per item is ${price} credits`;
  // Change code above this line
  return message;
};
```

Theory | **Task and tests** | Result — Done ← Assignment 10/36 →

The function `makeMessage(name, price)` composes and returns a message about the purchase. It declares two parameters, the values of which will be set during its call.

- `name` - product name
- `price` - product price

Modify the function code so that the `message` variable contains the string `"You picked <product name>, price per item is <product price> credits"`, where `<product name>` and `<product price>` are values of the parameters `name` and `price`. Use template string syntax.

Attention

Note that there are no calls to the `makeMessage` function in the code. From this task onwards, we will call your functions ourselves and check how they work. You will see the result of our checks in the `Results` block under the code editor.

TESTS

- The function `makeMessage(name, price)` is declared
- The call `makeMessage('Radar', 6150)` returns `"You picked Radar, price per item is 6150 credits"`

Editor javascript:

```
function calculateTotalPrice (orderedQuantity, pricePerItem) {
  // Change code below this line
  const totalPrice = orderedQuantity * pricePerItem;

  // Change code above this line
  return totalPrice;
};
```

Theory | **Task and tests** | Result — Done ← Assignment 11/36 →

The `calculateTotalPrice` function calculates and returns the total price based on the ordered quantity and the price per item. It takes two parameters, the values of which will be assigned during its call.

- `orderedQuantity` - the number of units of a product in the order
- `pricePerItem` - the price per item

Modify the function code to assign the total price, which is obtained by multiplying the ordered quantity by the price per item, to the `totalPrice` variable.

TESTS

- The function `calculateTotalPrice (orderedQuantity, pricePerItem)` is declared
- The call `calculateTotalPrice(5, 100)` returns `500`
- The call `calculateTotalPrice(8, 60)` returns `480`
- The call `calculateTotalPrice(3, 400)` returns `1200`
- The call `calculateTotalPrice(1, 3500)` returns `3500`

Editor javascript:

```
function makeOrderMessage(orderedQuantity, pricePerDroid, deliveryFee) {
  // Change code below this line
const totalPrice = orderedQuantity * pricePerDroid + deliveryFee;
const message = `You ordered droids worth ${totalPrice} credits. Delivery
(${deliveryFee} credits) is included in total price.`

  // Change code above this line
  return message;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 12/36 →

The `makeOrderMessage(orderedQuantity, pricePerDroid, deliveryFee)` function composes and returns a repair droid purchase message. It declares three parameters, the values of which will be set during its call.

- `orderedQuantity` - number of droids in the order
- `pricePerDroid` - price per droid
- `deliveryFee` - delivery fee

Modify the function code so that it returns an order message in the format `"You ordered droids worth <total price> credits. Delivery (<delivery fee> credits) is included in total price."`. Don't forget about the delivery fee when calculating the total price.

TESTS

- The function `makeOrderMessage(orderedQuantity, pricePerDroid, deliveryFee)` is declared
- The call `makeOrderMessage(2, 100, 50)` returns `"You ordered droids worth 250 credits. Delivery (50 credits) is included in total price."`
- The call `makeOrderMessage(4, 300, 100)` returns `"You ordered droids worth 1300 credits. Delivery`

Editor javascript:

```
function isAdult(age) {
  // Change code below this line
  const passed = age >= 18;

  // Change code above this line
  return passed;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 13/36 →

The `isAdult` function declares one `age` parameter, the value of which will be set when it is called. Set the variable `passed` to an expression for checking the age of the user. A person is considered an adult at the age of 18 or over.

TESTS

- The `isAdult(age)` function is declared
- The test expression uses the `>=` operator
- The call `isAdult(20)` returns `true`
- The call `isAdult(14)` returns `false`
- The call `isAdult(8)` returns `false`
- The call `isAdult(37)` returns `true`

Editor javascript:

```
function isValidPassword(password) {
  const SAVED_PASSWORD = 'jqueryismyjam';
  // Change code below this line
  const isMatch = password === SAVED_PASSWORD;

  // Change code above this line
  return isMatch;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 14/36 →

The `isValidPassword(password)` function checks the equality of the stored and entered passwords and returns the result of the check - boolean `true` or `false`. The variable `SAVED_PASSWORD` stores the value of the previously saved password. The entered password is passed to the `password` parameter.

Assign to the variable `isMatch` an expression for checking the equality of the previously entered and saved passwords. The result of the test expression must be `true` if the values match, and `false` if not.

TESTS

- The function `isValidPassword(password)` is declared
- The operator `===` is used in the password test expression
- The call `isValidPassword("mangodab3st")` returns `false`
- The call `isValidPassword("kiwirul3z")` returns `false`
- The call `isValidPassword("jqueryismyjam")` returns `true`

Editor javascript:

```
function checkAge(age) {
  let message;

  if (age >= 18) { // Change this line
    message = 'You are an adult';
  } else {
    message = 'You are a minor';
  }

  return message;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 15/36 →

Add the expression for checking the age of the user, the value of the `age` parameter, to the condition for the `if` statement.

- If the user is an adult, the `if` block should be executed and the string `"You are an adult"` is written to the `message` variable.
- Otherwise, the `else` block must be executed and the line `"You are a minor"` is written.

TESTS

- The function `checkAge(age)` is declared
- The `>=` operator is used in the age test expression
- The call `checkAge(20)` returns `"You are an adult"`
- The call `checkAge(8)` returns `"You are a minor"`
- The call `checkAge(14)` returns `"You are a minor"`
- The call `checkAge(38)` returns `"You are an adult"`

Editor javascript:

```
function checkStorage(available, ordered) {
  let message;
  // Change code below this line
if (ordered > available) {
  message = "Not enough goods in stock!";
} else {
    message = "Order is processed, our manager will contact you."
  }
  // Change code above this line
  return message;
}

```

Theory | **Task and tests** | Result — Done — Assignment 16/36

The `checkStorage(available, ordered)` function checks the order capability and returns a result message. It declares two parameters, the values of which will be set during its call:

- `available` - the total number of products in the warehouse
- `ordered` - units of goods in the order

Using branches, modify the function code so that:

- If the order contains a number exceeding the number of goods in stock, the string `"Not enough goods in stock!"` Is written to the `message` variable.
- Otherwise, the string `"Order is processed, our manager will contact you."` is written.

TESTS

- The function `checkStorage(available, ordered)` is declared.
- The call `checkStorage(100, 50)` returns `"Order is processed, our manager will contact you."`
- The call `checkStorage(100, 130)` returns `"Not enough goods in stock!"`
- The call `checkStorage(200, 20)` returns `"Order is processed, our manager will contact you."`

Editor javascript:

```
let a = 5;
let b = 10;
let c = 15;
let d = 20;

// Change code below this line
a += 2;
b -= 4;
c *= 3;
d /= 10;

```

Theory | **Task and tests** | Result — Done — Assignment 17/36

Replace the expression with the standard math operators with the combined assignment operator with addition, subtraction, multiplication, and division.

TESTS

- The value of the variable `a` is equal to `7`
- Operator `+=` used
- The value of the variable `b` is equal to `6`
- Operator `-=` used
- The value of the variable `c` is equal to `45`
- Operator `*=` used
- The value of the variable `d` is equal to `2`
- Operator `/=` used

Editor javascript:

```
function makeTransaction(pricePerDroid, orderedQuantity, customerCredits) {
  let message;
  // Change code below this line
const totalPrice = pricePerDroid * orderedQuantity;
  if (totalPrice > customerCredits) {
    message = "Insufficient funds!";
  } else {
    message = `You ordered ${orderedQuantity} droids, you have ${customerCredits - totalPrice} credits left`;
  }
  // Change code above this line
  return message;
}

```

Theory | **Task and tests** | Result — Done — Assignment 18/36

The repair droid sales station is ready, all that remains is to write the software for the sales department.

The `makeTransaction(pricePerDroid, orderedQuantity, customerCredits)` function executes the transaction for the sale of droids and returns a message about the result of the operation. It declares three parameters, the values of which will be set during its call:

- `pricePerDroid` - price per droid
- `orderedQuantity` - number of ordered droids
- `customerCredits` - amount of funds on the customer's account

Add the following functionality to it:

- Declare the variable `totalPrice` to store the total amount of the order and assign it an expression for calculating this amount.
  - Add a check to find out if the client can pay for the order:
  - if the amount to be paid exceeds the number of credits on the client's account, write the string `"Insufficient funds!"` to the `message` variable;

Editor javascript:

```
function checkPassword(password) {
  const ADMIN_PASSWORD = 'jqueryismyjam';
  let message;

  if (password === null) { // Change this line
    message =  'Canceled by user!';
  } else if (password === ADMIN_PASSWORD) { // Change this line
    message = 'Welcome!';
  } else {
    message = 'Access denied, wrong password!';
  }

  return message;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 19/36 →

The `checkPassword(password)` function gets the user's password in the `password` parameter, checks it against the administrator password in the variable `ADMIN_PASSWORD` and returns the message about the result of the comparison, stored in the variable `message`.

- If the value of the `password` parameter is `null`, then the user canceled the operation and the string `"Canceled by user!"` is written to the `message` variable.
- If the value of the `password` parameter is the same as the value of `ADMIN_PASSWORD`, the string `"Welcome!"` is assigned to the variable `message`.
- If none of the previous conditions are met, the string `"Access denied, wrong password!"` is written to the `message` variable.

TESTS

- The function `checkPassword(password)` is declared
- The call `checkPassword("mangohackzor")` returns `"Access denied, wrong password!"`
- The call `checkPassword(null)` returns `"Canceled by user!"`
- The call `checkPassword("polyhax")` returns `"Access denied, wrong password!"`

Editor javascript:

```
function checkStorage(available, ordered) {
  let message;
  // Change code below this line
if (ordered === 0) {
  message = "There are no products in the order!"
} else if (ordered > available) {
  message = "Your order is too large, there are not enough items in stock!"
} else {
  message = "The order is accepted, our manager will contact you"
}
  // Change code above this line
  return message;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 20/36 →

The `checkStorage(available, ordered)` function checks the checkout capability and returns a result message. It declares two parameters, the values of which will be set during its call.

- `available` - available quantity of goods in the warehouse
- `ordered` - units of goods in the order

Using branches, add the function code so that:

- If there are no products in the order yet, that is, the value of the `ordered` parameter is equal to 0, the string `"There are no products in the order!"` is assigned to the `message` variable.
- If there are more goods in the order than there are available in the warehouse, then the string `"Your order is too large, there are not enough items in stock!" is assigned to the variable` message`.
- Otherwise, the string `"The order is accepted, our manager will contact you"` is assigned to the variable `message`.

TESTS

- The function `checkStorage(available, ordered)` is declared

Editor javascript:

```
function isNumberInRange(start, end, number) {
  const isInRange = number >= start && number <= end; // Change this line

  return isInRange;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 21/36 →

The `isNumberInRange(start, end, number)` function checks if a number is in the range. It declares three parameters, the values of which will be set during its call:

- `number` - the number; the function check if the number is in the range
- `start` - start of a number range
- `end` - end of a number range

Assign to the variable `isInRange` an expression to test for the occurrence of `number` in a numeric range from `start` to `end`. That is, the number must be greater than or equal to `start` and less than or equal to `end`. The result of the test expression will be boolean `true` or `false`.

TESTS

- The function `isNumberInRange(start, end, number)` is declared
- The `&&` operator is used in the test expression
- The call `isNumberInRange(10, 30, 17)` returns `true`
- The call `isNumberInRange(10, 30, 5)` returns `false`

Editor javascript:

```
function checkIfCanAccessContent(subType) {
  const canAccessContent = subType === "pro" || subType === "vip"; // Change this line

  return canAccessContent;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 22/36 →

The `checkIfCanAccessContent(subType)` function checks if the user can access the content. The check is carried out by the type of subscription. Only users with a `pro` or `vip` subscription can be accessed.

Set the variable `canAccessContent` to a subscription test expression. If the value of the `subType` parameter is equal to the strings `"pro"` or `"vip"`, the user will have access. The result of the test expression will be boolean `true` or `false`.

TESTS

- The function `checkIfCanAccessContent(subType)` is declared
- The operator `||` is used in the test expression
- The call `checkIfCanAccessContent("pro")` returns `true`
- The call `checkIfCanAccessContent("starter")` returns `false`
- The call `checkIfCanAccessContent("vip")` returns `true`
- The call `checkIfCanAccessContent("free")` returns `false`

Editor javascript:

```
function isNumberNotInRange(start, end, number) {
  const isInRange = number >= start && number <= end;
  const isNotInRange = !isInRange; // Change this line

  return isNotInRange;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 23/36 →

The `isNumberNotInRange(start, end, number)` function checks if a number is in a range. That is, the number must be less than or equal to `start` and greater than or equal to `end`. The result of the test expression will be boolean `true` or `false`.

It declares three parameters, the values of which will be set during its call:

- `number` - a number whose occurrence is checked to be not in the range
- `start` - start of a numeric range
- `end` - end of a numeric range

Assign to the variable `isNotInRange` the expression of the inverse of the variable `isInRange` value using the operator `!`.

TESTS

- The function `isNumberNotInRange(start, end, number)` is declared
- The expression used the `!` operator
- The call `isNumberNotInRange(10, 30, 17)` returns `false`

Editor javascript:

```
function getDiscount(totalSpent) {
  const BASE_DISCOUNT = 0;
  const BRONZE_DISCOUNT = 0.02;
  const SILVER_DISCOUNT = 0.05;
  const GOLD_DISCOUNT = 0.1;
  let discount;
  // Change code below this line
if (totalSpent >= 50000) {
  discount = GOLD_DISCOUNT
} else if (totalSpent >= 20000 && totalSpent < 50000) {
  discount = SILVER_DISCOUNT
} else if (totalSpent >= 5000 && totalSpent < 20000) {
  discount = BRONZE_DISCOUNT
} else {
  discount = BASE_DISCOUNT
}
  // Change code above this line
  return discount;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 24/36 →

The `getDiscount(totalSpent)` function determines the discount value depending on the total amount of money spent (the `totalSpent` parameter) in the store for the entire time (affiliate program). The discount is written to the variable `discount` and returns from the function as a result of its operation.

Using branches and logical operators, complete the function code.

- If spent from `50,000` (inclusive) or more credits - discount `10%` (golden partner)
- If spent from `20,000` (inclusive) to `50,000` credits - discount `5%` (silver partner)
- If spent from `5000` (inclusive) to `20,000` credits - discount `2%` (bronze partner)
- If spent less than `5000` credits - discount `0` (basic partner)

Discount values for each level are stored in the constants of the same name. `BASE_DISCOUNT`, `BRONZE_DISCOUNT`, `SILVER_DISCOUNT` and `GOLD_DISCOUNT`.

TESTS

- The function `getDiscount(totalSpent)` is declared

Editor javascript:

```
function checkStorage(available, ordered) {
  let message;
  // Change code below this line
message = ordered > available ? "Not enough goods in stock!" : "The order is accepted, our manager will contact you"

  // Change code above this line
  return message;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 25/36 →

Refactor the solution to the "Warehouse" problem by replacing the `if...else` statement with a ternary operator.

TESTS

- The function `checkStorage(available, ordered)` is declared.
- Ternary operator is used.
- The call `checkStorage(100, 50)` returns `"The order is accepted, our manager will contact you"`
- The call `checkStorage(100, 130)` returns `"Not enough goods in stock!"`
- The call `checkStorage(200, 20)` returns `"The order is accepted, our manager will contact you"`
- The call `checkStorage(200, 150)` returns `"The order is accepted, our manager will contact you"`
- The call `checkStorage(150, 180)` returns `"Not enough goods in stock!"`

Editor javascript:

```
function checkPassword(password) {
  const ADMIN_PASSWORD = "jqueryismyjam";
  let message;
  // Change code below this line
message = password === ADMIN_PASSWORD ? "Access is allowed" : "Access denied, wrong password!"
  // Change code above this line
  return message;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 26/36 →

The `checkPassword(password)` function compares the password passed to it (the `password` parameter) with the saved administrator password (the `ADMIN_PASSWORD` constant) and returns a string with a message about the result.

Using the ternary operator, modify the function so that:

- If the values of `password` and `ADMIN_PASSWORD` are the same, the variable `message` is set to the string `"Access is allowed"`.
- Otherwise, set the `message` to the string "Access denied, wrong password!"

TESTS

- The function `checkPassword(password)` is declared
- Ternary operator is used
- The call `checkPassword("jqueryismyjam")` returns `"Access is allowed"`
- The call `checkPassword("angul4r1sl1f3")` returns `"Access denied, wrong password!"`
- The call `checkPassword("r3actsux")` returns `"Access denied, wrong password!"`

Editor javascript:

```
function getSubscriptionPrice(type) {
  let price;
  // Change code below this line

  switch (type) { // Change this line
    case "starter": // Change this line
      price = 0; // Change this line
      break;

    case "professional": // Change this line
      price = 20; // Change this line
      break;

    case "organization": // Change this line
      price = 50; // Change this line
      break;
  }

  // Change code above this line
  return price;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 27/36 →

The `getSubscriptionPrice(type)` function gets a string with the user's subscription type (the `type` parameter), checks it against the three possible types of monthly subscriptions, and returns the price stored in the `price` variable.

If the value of the `type` parameter is a string:

- `"starter"` - subscription price is `0` credits.
- `"professional"` - subscription price is `20` credits.
- `"organization"` - subscription price is `50` credits.

Initially, the body of the function had an `if...else` statement that looked like this.

```
if (type === "starter") {
  price = 0;
} else if (type === "professional") {
  price = 20;
} else if (type === "organization") {
  price = 50;
}
```

After refactoring, the `if..else` statement was replaced with `switch`. Complete the `switch` state-

Editor javascript:

Theory | **Task and tests** | Result — Done ← Assignment 28/36 →

```
function checkPassword(password) {
  const ADMIN_PASSWORD = "jqueryismyjam";
  let message;
  // Change code below this line
switch (password) {
    case null:
    message = "Canceled by user!";
    break;

  case ADMIN_PASSWORD:
    message = "Welcome!";
    break;

  default:
    message = "Access denied, wrong password!";
}
  // Change code above this line
  return message;
}

```

The `checkPassword(password)` function takes the password in the `password` parameter, checks it against the administrator password in the `ADMIN_PASSWORD` variable, and returns a message about the result of the comparison, which is stored in the `message` variable.

If the value of the parameter `password`:

- is equal to `null`, it means the user canceled the operation and the string `"Canceled by user!"` is written to `message`.
- matches the value of `ADMIN_PASSWORD`, the string `"Welcome!"` is assigned to the variable `message`.
- does not satisfy any of the previous conditions, the string `"Access denied, wrong password!"` is written to the `message` variable.

Refactor the code by replacing the `if..else` statement with a `switch` statement and don't forget about the `default` block (analogous to else).

TESTS

- The function `checkPassword(password)` is declared
- The call `checkPassword("mangohackzor")` returns `"Access denied, wrong password!"`

Editor javascript:

Theory | **Task and tests** | Result — Done ← Assignment 29/36 →

```
function getShippingCost(country) {
  let message;
  let price = '';
  // Change code below this line
switch (country) {
    case "China":
    price = 100;
    message = `Shipping to ${country} will cost ${price} credits`;
    break;

    case "Chile":
    price = 250;
    message = `Shipping to ${country} will cost ${price} credits`;
    break;

    case "Australia":
    price = 170;
    message = `Shipping to ${country} will cost ${price} credits`;
    break;

    case "Jamaica":
    price = 120;
    message = `Shipping to ${country} will cost ${price} credits`;
    break;

    default:
    message = "Sorry, there is no delivery to your country";
}
  // Change code above this line
  return message;
}

```

The `getShippingCost(country)` function should check the possibility of product delivery to the user's country (the country parameter) and return a message about the result, which stored in the `message` variable. Be sure to use the `switch` statement.

The format of the returned string is `"Shipping to <country> will cost <price> credits"`, where the corresponding values must be substituted for `<country>` and `<price>`.

List of countries and delivery fee:

- `China` - 100 credits
- `Chile` - 250 credits
- `Australia` - 170 credits
- `Jamaica` - 120 credits

The list shows that delivery is not available everywhere. If the specified country is not in the list, then the function should return the string `"Sorry, there is no delivery to your country"`

TESTS

- The function `getShippingCost(country)` is declared

Editor javascript:

Theory | **Task and tests** | Result — Done ← Assignment 30/36 →

```
function getNameLength(name) {
  const message = `Name ${name} is ${name.length} characters long`; // Change this line

  return message;
}

```

The `getNameLength(name)` function takes a name (the name parameter) and returns a string specifying its length. Supplement the template string in the `message` variable with the length of the string from the `name` parameter.

TESTS

- The function `getNameLength(name)` is declared
- The function `getNameLength("Poly")` call returns `"Name Poly is 4 characters long"`
- The function `getNameLength("Harambe")` call returns `"Name Harambe is 6 characters long"`
- The function `getNameLength("Billy")` call returns `"Name Billy is 5 characters long"`
- The function `getNameLength("Joe")` call returns `"Name Joe is 3 characters long"`

Editor javascript:

```
const courseTopic = "JavaScript essentials";
// Change code below this line

const courseTopicLength = courseTopic.length;
const firstElement = courseTopic[0];
const lastElement = courseTopic[courseTopic.length - 1];

// Change code above this line

```

Theory | **Task and tests** | Result — Done ← Assignment 31/36 →

Complete the code by assigning to the declared variables the expressions of accessing the corresponding elements or properties of the string in the `course` variable.

- `courseTopicLength` - string length.
- `firstElement` - the first element of the string.
- `lastElement` - the last element of the string.

TESTS

- The variable `courseTopic` is declared
- The value of the `courseTopic` variable is the string `"JavaScript essentials"`
- The variable `courseTopicLength` is declared
- The value of the `courseTopicLength` variable is the number `21`
- The variable `firstElement` is declared
- The value of the `firstElement` variable is the string `"J"`

Editor javascript:

```
function getSubstring(string, length) {
  const substring = (string.slice(0, length)); // Change this line

  return substring;
}
```

Theory | **Task and tests** | Result — Done ← Assignment 32/36 →

The `getSubstring(string, length)` function takes a string and returns the substring from the beginning character to the value passed in the parameter `length`. It declares two parameters, the values of which will be set during its call:

- `string` - original string
- `length` - the number of characters to slice starting from the beginning of the string for the resulting substring

Assign to the variable `substring` an expression for creating a substring of `length` characters from the beginning of the `string`.

TESTS

- The function `getSubstring(string, length)` is declared
- The function `getSubstring("Hello world", 3)` call returns `"Hel"`
- The function `getSubstring("Hello world", 6)` call returns `"Hello"`
- The function `getSubstring("Hello world", 8)` call returns `"Hello wo"`

Editor javascript:

```
function formatMessage(message, maxLength) {
  let result;
  // Change code below this line
if (message.length <= maxLength) {
  result = message;
} else {
  slisedMessage = message.slice(0, maxLength);
  result = `${slisedMessage}...`
}
  /// Change code above this line
  return result;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 33/36 →

The `formatMessage(message, maxLength)` function accepts a string (`message` parameter) and checks its length against the specified maximum length (`maxLength` parameter).

- If the length of the string is equal to or shorter than `maxLength`, the function returns the original string as it is.
- However, if the length exceeds `maxLength`, the function truncates the string to `maxLength` characters and adds an ellipsis (`'...'`) at the end, then returns the truncated version.

TESTS

- The function `formatMessage(message, maxLength)` is declared
- The function `formatMessage("Curabitur ligula sapien", 16)` call returns `"Curabitur ligula..."`
- The function `formatMessage("Curabitur ligula sapien", 23)` call returns `"Curabitur ligula sapien"`
- The function `formatMessage("Vestibulum facilisis purus nec", 20)` call returns `"Vestibulum facilisis..."`
- The function `formatMessage("Vestibulum facilisis purus nec", 30)` call returns `"Vestibulum facilisis purus nec"`

Editor javascript:

```
function normalizeInput(input) {
  const normalizedInput = input.toLowerCase(); // Change this line

  return normalizedInput;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 34/36 →

The `normalizeInput(input)` function takes a string (input parameter) and returns the same string, but in lowercase. Assign to the variable `normalizedInput` an expression to create a lowercase string from the `input` parameter.

TESTS

- The function `normalizeInput(input)` is declared
- The function `normalizeInput("Hello world")` call returns `"hello world"`
- The function `normalizeInput("This ISN'T SpaM")` call returns `"this isn't spam"`
- The function `normalizeInput("Big SALE")` call returns `"big sale"`

Editor javascript:

```
function checkForName(fullName, name) {
  const result = fullName.includes(name); // Change this line
  return result;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 35/36 →

The `checkForName(fullname, name)` function takes two parameters and returns boolean `true` or `false` - the result of checking whether the substring `name` is in the string `fullname`.

- `fullname` - full name consisting of two words (first and last name) separated by a space
- `name` - name to check for occurrence in the `fullname` parameter

Assign to the variable `result` an expression for checking the occurrence of the name (the `name` parameter), in the full name (the `fullname` parameter). Let the function strictly relate to the case of letters (case-sensitive), that is, "Petya" and "petya" are different names for it.

TESTS

- The function `checkForName(fullname, name)` is declared.
- The function `checkForName("Egor Kolbasov", "Egor")` call returns `true`
- The function `checkForName("Egor Kolbasov", "egor")` call returns `false`
- The function `checkForName("Egor Kolbasov", "egOr")` call returns `false`
- The function `checkForName("Egor Kolbasov", "Zhenya")` call returns `false`

Editor javascript:

```
function checkForSpam(message) {
  let result;
  // Change code below this line
normilizedMessage = message.toLowerCase();

check = normilizedMessage.includes("spam") || normilizedMessage.includes("sale");
  result = check;
  // Change code above this line
  return result;
}

```

Theory | **Task and tests** | Result — Done ← Assignment 36/36

The `checkForSpam(message)` function is designed to analyze a string (message parameter) and determine if it contains any forbidden words, namely `'spam'` or `'sale'`. It performs a case-insensitive search, meaning variations in capitalization (e.g., `'SPAM'` or `'sAlE'`) are considered.

- If any forbidden word is found in the string (`message`), the function has to return `true`.
- On the other hand, if no forbidden words are detected, the function has to return `false`.

TESTS

- The function `checkForSpam(message)` is declared.
- The function `checkForSpam("Latest technology news")` call returns `false`
- The function `checkForSpam("JavaScript weekly newsletter")` call returns `false`
- The function `checkForSpam("Get best sale offers now!")` call returns `true`
- The function `checkForSpam("Amazing SalE, only tonight!")` call returns `true`
- The function `checkForSpam("Trust me, this is not a spam message")` call returns `true`
- The function `checkForSpam("Get rid of sPaM emails. Our book in on sale!")` call returns `true`