Solution 1:

```
function outerFunction(param) {

const outerVariable = 'I am from outerFunction';

function innerFunction() {

console.log('Parameter from outerFunction:', param);
console.log('Variable from outerFunction:', outerVariable);
}
return innerFunction;
}

const myInnerFunction = outerFunction('Hello, world!');


myInnerFunction();
```

Solution 2:

```
function matchesPattern(pattern, str) {
const regex = new RegExp(pattern);


 return regex.test(str);
}

 const tests = [
 { pattern: '\\d+', str: '123abc', expected: true },
 { pattern: '\\d+', str: 'abc', expected: false },
 { pattern: '^Hello', str: 'Hello, world!', expected: true },
 { pattern: 'world!$', str: 'Hello, world!', expected: true },
 { pattern: 'world$', str: 'Hello, world!', expected: false },
 { pattern: '[A-Za-z]+', str: 'abc123', expected: true },
 { pattern: '[A-Za-z]+', str: '123', expected: false },
 { pattern: '\\bword\\b', str: 'The word is here.', expected: true },
 { pattern: '\\bword\\b', str: 'The password is here.', expected: false }
];

tests.forEach(({ pattern, str, expected }) => {
 const result = matchesPattern(pattern, str);
 console.log(`Pattern: /${pattern}/ | String: "${str}" | Expected: ${expected} | Result: ${result}`);
});
```

Solution 3:

```
function findMatches(pattern, str) {

const regex = new RegExp(pattern, 'g');
```

```
  const matches = str.match(regex);
  return matches || [];
 }


  const tests = [
  { pattern: '\\d', str: '123abc456', expected: ['1', '2', '3', '4', '5', '6'] },
  { pattern: '[A-Z]', str: 'Hello World!', expected: ['H', 'W'] },
  { pattern: '[a-z]', str: 'Hello World!', expected: ['e', 'l', 'l', 'o', 'o', 'r', 'l', 'd'] },
  { pattern: '[!@#$%^&*(),.?":{}|<>]', str: 'Hello World! @2024$', expected: ['!', '@', '$'] },
  { pattern: '\\s', str: 'Hello World! @2024$', expected: [' ', ' ', ' ', ' '] }
  ];

  tests.forEach(({ pattern, str, expected }) => {
  const result = findMatches(pattern, str);
  console.log(`Pattern: /${pattern}/ | String: "${str}" | Expected: $
{JSON.stringify(expected)}     | Result: ${JSON.stringify(result)}`);
});
```

Solution 4:

```
function extractWithGroups(pattern, str) {
 const regex = new RegExp(pattern);
 const match = regex.exec(str);
 return match ? match.slice(1) : null;
}

 const tests = [
  {
   pattern: '(\\d{2})/(\\d{2})/(\\d{4})',
   str: 'The date is 15/08/2024.',
   expected: ['15', '08', '2024']
  },
  {
   pattern: '(\\d{4})-(\\d{2})-(\\d{2})',
   str: 'The date is 2024-08-15.',
   expected: ['2024', '08', '15']
  },
  {
   pattern: '(\\d{1,2})/(\\d{1,2})/(\\d{2,4})',
   str: 'My birthday is 5/9/99.',
   expected: ['5', '9', '99']
  },
  {
   pattern: '(\\d{2})-(\\d{2})-(\\d{2})',
   str: 'The expiration date is 31-12-25.',
   expected: ['31', '12', '25']
  }
];
```

```
tests.forEach(({ pattern, str, expected }) => {
  const result = extractWithGroups(pattern, str);
  console.log(`Pattern: /${pattern}/ | String: "${str}" | Expected: ${JSON.stringify(expected)} |
Result: ${JSON.stringify(result)}`);
});
```

Solution 5:

```
   function getDeliveryTime(packageType) {
   let deliveryTime;

  switch (packageType) {
    case 'standard':
      deliveryTime = '3-5 days';
      break;
    case 'express':
      deliveryTime = '1-2 days';
      break;
    case 'overnight':
      deliveryTime = 'Next day';
      break;
    default:
      deliveryTime = 'Invalid package type';
    }

     console.log(`Estimated delivery time for ${packageType} package: ${deliveryTime}`);
     }

    getDeliveryTime('standard');
    getDeliveryTime('express');
    getDeliveryTime('overnight');
    getDeliveryTime('priority');
```