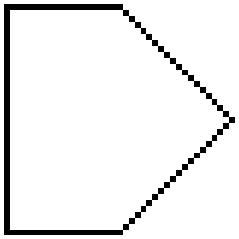
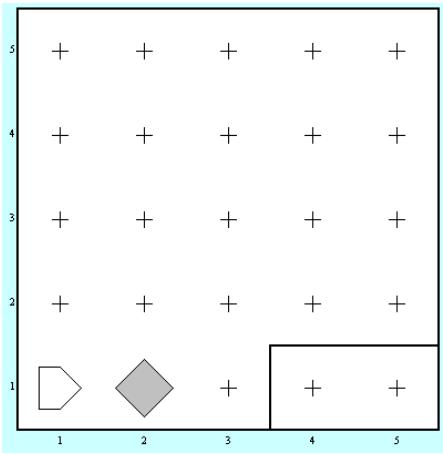
რობოტი კარელი

პროგრამირების მარტივად ათვისებაში დაგვეხმარება რობოტი კარელი. კარელი რობოტია, რომელმაც ძალიან ცოტა რაღაცის გაკეთება იცის, მაგრამ ჩვენ ბევრი რამის სწავლაში დაგვეხამრება.



კარელის სამყარო

კარელი მარტივ სამყაროში ცხოვრობს. მისი სამყარო მართხკუთხა ფორმისაა, აქვს ქუჩები და გამზირები. ქუჩები განლაგებულია ჰორიზონტალურად - დაკარელითიდან აღმოკარელითის მიმართულებით, გამზირები კი ვერტიკალურად სამხრეთიდან ჩრდილოეთის მიმართულებით. კარელის კოორდინატები შეიძლება იყოს ნებისმიერი ქუჩისა და გამზირის გადაკვეთაზე და იყურებოდეს ნებისმიერი მიმართულებით (აღმოკარელითი, სამხრეთი, დაკარელითი, ჩრდილოეთი). სამყაროს მაგალითი შეგიძლიათ იხილოთ ქვევით. ამ მაგალითზე კარელი იმყოფება პირველი ქუჩის პირველ გამზირზე.



როგორც ვხედავთ, სამყაროში, კარელის გარდა, სხვა რაღაცეებიც არის. კარელსის წინ არსებული ობიექტი არის ბრილიანტი. კარელს ბრილიანტის აღმოჩენა შეუძლია მხოლოდ მაშინ, თუ ის იგივე ქუჩაზე და გამზირზეა, რომელზეც ბრილიანტია.
სამყაროში გვაქვს კედლებიც. კედელი კარელისთვის ბარიერს წარმოადგენს, რადგან მას არ შეუძლია კედელში გავლა, შესაბამისად, მას უნდა შემოუაროს. კარელს ასევე აქვს ჩანთა, სადაც ბრილიანტები აქვს ჩალაგებული. ჩანთაში ნებისმიერი რაოდენობის ბრილიანტი შეიძლება იყოს. კარელის სამყარო ყოველთვის კედლებით არის შემოფარგლული, თუმცა, შეიძლება მას სხვადასხვა განზომილება ჰქონდეს, ბრილიანტები შეიძლება სხვადასხვა ადგილებში იდოს, კედლებიც შეიძლება სხვადასხვა ქუჩებზე და გამზირებზე გვქონდეს, იმის მიხედვით, თუ რა პრობლემის გადაჭრას ცდილობს კარელი.

რა შეუძლია კარელის გააკეთოს?

კარელის შეუძლია გარკვეული ბრძანებების შესრულება, ესენია:
move() - წინ გადაადგილება. კარელი ვერ გადაადგილდება თუ მის წინ კედელია.
turnLeft() - მარცხნივ მოხვევა. კარელი მოტრიალდება მარცხნივ 90 გრადუსით.
pickBeeper() - ბრილიანტის აღება. კარელი ვერ აიღებს ბრილიანტს, თუ ბრილიანტი არ დევს იმავე ადგილზე, სადაც თვითონ კარელი იმყოფება.
putBeeper() - ბრილიანტის დადება. კარელი ვერ შეძლებს ბრილიანტის დადებას, თუკი მას ჩანთაში არ აქვს ბრილიანტი.

გახსნილი და დახურული ფრჩხილი () უბრალოდ სინტაქსის ელემენტია და აღმნიშვნავს ფუნქციას პროგრამირების ენა ჯავასკრიპტში. ფუნქციის საშუალებით კარელის შეუძლია ბრძანებების შესრულება. move(), turnLeft(), pickBeeper(), putBeeper(), უკვე არსებული ფუნქციებია, რომელთა შესულებაც კარელის შეუძლია.

ალგორითმი და პირველი პროგრამა

რა არის ალგორითმი? ალგორითმი არის ამოცანის გადაჭრისთვის საჭირო ინსტრუქციების მიმდევრობა, რომელიც ადამიანისთვის გასაგებ ენაზეა დაწერილი. იმისათვის, რომ ალგორითმიდან გამართული და მუშა პროგრამა მივიღოთ კოდის დაწერა მოგვიწევს.
დავუშვათ, გვინდა, რომ რობოტმა კარელმა აიღოს მეორე გამზირის პირველ ქუჩაზე არსებული ბრილიანტი და დადოს მეოთხე გამზირის მეორე ქუჩაზე. ამის გაკეთება ბევრი გზით არის შესაძლებელი. თითოეული გზა განსხვავებული ალგორითმია. მაგალითად, ერთ-ერთი ალგორითმია

წავიდეთ წინ, ავიღოთ ბრილიანტი, წავიდეთ წინ, ავიდეთ ზემოთ, წავიდეთ წინ, დავდოთ ბრილიანტი.
არსებობს კარგი და ცუდი ალგორითმები. იგივე ამოცანის გადაჭრა შეგვეძლო ცუდი ალგორითმითაც. მაგალითად სანამ ბრილიანტს აიღებს კარელი მანამ შეიძლება მთელი სამყარო შემოიაროს, მხოლოდ შემდეგ აიღოს ბრილიანტი და დადოს მითითებულ ადგილას. ეს ალგორითმიც ამოცანას გადაჭრის, თუმცა, ამისათვის კარელის გაცილებით მეტი სიარული მოუწევს. შესაბამისად, ალგორითმები შეგვიძლია ერთმანეთს შევადაროთ იმის მიხედვით, თუ რომელი უფრო მეტ ენეგიას და დროს ხარჯავს. ცხადია, ჯობია ისეთი ალგორითმით გავაკეთოთ საქმე, რომლითაც დანახარჯი ნაკლები იქნება.
მოდით, დავწეროთ დასმული ამოცანისთვის პროგრამული კოდი. თუ დააკვირდებით ეს პროგრამა განსხვავებული ალგორითმით აკეთებს საქმეს.

```
move();
pickBeeper();
turnLeft();
move();
turnLeft();
turnLeft();
turnLeft();
move();
move();
putBeeper();
```

ახალი ფუნქციის შექმნა

კარელს არსებული ბრძანებების გარდა შეიძლება ახალი ბრძანებებიც ვასწავლოთ. ამ ეტაპზე კარელმა, როგორც სოციალისტური შრომის გმირმა იცის მხოლოდ მარცხნივ მოხვევა. ამიტომ, კარგი იქნება, თუ გავუფართოებთ ცოდნას და მარჯვნივ მოხვევასაც ვასწავლით. ამისათვის, უნდა აღვწეროთ ახალი ფუნქცია. ფუნქციის აღწერა ორი ნაწილისგან შედგება ფუნქციის თავი, სადაც ფუნქციის სახელი იწერება და ფუნქციის ტანი, სადაც იწერება თუ რა უნდა გააკეთოს კარელმა ამ ფუქციის გამოძახების შემდეგ. აღსანისნავია, რომ ფუნქციის ტანი აუცილებლად უნდა იყოს ფიგურულ ფრჩხილებში მოქცეული, რათა ჯავასკრიპტის ინტერპრეტატორმა შეძლოს გარჩევა, თუ სად იწყება და მთავრდება ფუნქციის განმარტება. მარჯვნივ მოხვევის ფუნქციის მაგალითი:

```
function turnRight(){
    turnLeft();
    turnLeft();
    turnLeft();
}
```

function აღნიშნავს ფუნქციის აღწერის დასაწყისს.
turnRight ჩვენ მიერ შერჩეული სახელია. სახელად, ცხადია, ნებისმიერი სხვა რამ შეგვეძლო შეგვერჩია მაგალითად “moukhvie_marjvniv” ან “marjvnisaken” თუმცა სტილისტურად turnRight უფრო ემთხვევა სხვა ფუნქციების სახელებს და ამიტომაც შევარჩიეთ. მრგვალი ფრჩხილები ამ სიტყვას აქცევს მეთოდად, ხოლო ფიგურულ ფრჩხილებში მოთავსებულია პროგრამის ტანი.

repeat ციკლი

ხშირად არის სიტუაცია, როდესაც გვინდა ბრძანებების რაღაც მიმდევრობა რაღაც სიხშირით გავიმეოროთ. იმის მაგივრად, რომ copy/paste გავაკეთოთ შეგვიძლია repeat ციკლი გამოვიყენოთ.
თავიდანვე მინდა აღვნიშნო, რომ copy/paste ძალიან ცუდი იდეაა პროგრამის წერის დროს. ამ დროს პროგრამაში ჩნდება ორ ან უფრო მეტ ადგილზე ერთიდაიგივე, გამეორებული კოდი. კოდის გამეორება არ არის სასურველი, ერთი იმიტომ, რომ ზედმეტია და არ არის საჭირო, მეორე იმიტომ რომ შეცდომების წყაროა. წარმოიდგინეთ სიტუაცია, როდესაც ერთიდაიგივე კოდი 10-ჯერ გაქვთ გადაკოპირებული და აღმოჩნდა, რომ ეს კოდი შესაცვლელია. უნდა ჩამოუაროთ ათივე ადგილს და ყველგან შეცვალოთ. თუკი რომელიმე ადგილი დაგავიწყდათ, თქვენი პროგრამა აღარ იქნება გამართული. იმ შემთხვევაში კი, როდესაც კოდის გამეორება არ გვაქვს ცვლილებებიც მარტივი შესატანია და შეცდომებსაც არ იწვევს.
მივუბრუნდეთ repeat ციკლს. დავუშვათ, კარელი 100x100 ზომის სამყაროში ცხოვრობს, ახლა არის პირველი გამზირის პირველ ქუჩაზე და უნდა აიღოს ბრილიანტი, რომელიც მდებარეობს მეასე გამზირის პირველ ქუჩაზე. ამისათვის, უკვე არსებული ცოდნითაც შეგვიძლია დავწეროთ მარტივი პროგრამა, რომელშიც ერთმანეთის ქვევით ეწერება 99 ცალი move(). ამის ნაცვლად შეგვიძლია გამოვიყენოთ repeat ციკლი სადაც მივუთითებთ, რომ ინსტრუქცია move()-ის გაკეთება გვინდა 99-ჯერ.

```
repeat(99){
    move();
}
```

reoeat ციკლი, ისევე როგორც ფუნქცია, შედგება ორი ნაწილისგან. repeat ციკლის თავისგან და repeat ციკლის ტანისგან. repeat ციკლის თავში შეგიძლიათ ჩაწეროთ ნებისმიერი რიცხვი და ინსტრუქციების ის მიმდევრობა, რაც repeat ციკლის ტანში გიწერიათ, გამეორდება იმდენჯერ, რა რიცხვსაც ჩაწერთ. გაითვალისწინეთ, რომ ფუნქციის მსგავსად repeat ციკლის ტანში არსებული ყველა ფუნქცია უნდა იყოს მოქცეული ფიგურულ ფრჩხილებში.

რა თქმა უნდა, repeat ციკლის ტანში შეიძლება არა ერთი, არამედ რამდენიმე ინსტრუქცია/ბრძანება ეწეროს. ამ შემთხვევაში არსებული ინსტრუქციები მეორდება მიმდევრობით (თითქოს ეს ინსტრუქციები ერთმანეთის ქვეშ დააკოპირეს). მაგალითად კოდი

```
repeat(4){
    move();
    turnLeft();
}
```

ნიშნავს არა იმას, რომ კარელმა ჯერ ოთხი move() უნდა შეასრულოს და შემდეგ ოთხი turnLeft(), არამედ ნიშანავს, ჯერ შეასრულოს move() მერე turnLeft() და ეს გაიმეოროს 4-ჯერ. რის შედეგადაც კარელი საწყის წერტილზე დაბრუნდება, თუკი ირგვლივ კედლები არ არის.

while ციკლი

for ციკლი წინა ლექციაზე ავხსენით. მისი გამოყენება კარგია მაშინ, როდესაც გარკვეული ბრძანებების მიმდევრობის გამეორება გვინდა და ვიცით, რამდენჯერ გვინდა გავიმეოროთ. ამ დროს ბრძანებების მიმდევრობას ვწერთ for ციკლის ტანში, for ციკლის თავში ვუთითებთ რაოდენობას (კონკრეტულ რიცხვს), თუ რამდენჯერ გვინდა გამეორება.

ხშირად არის სიტუაცია, როდესაც რაღაცის გამეორება გვინდა, მაგრამ ზუსტად არ ვიცით რამდენჯერ. ამ შემთხვევაში გამოგვადგება while ციკლი. მაგრამ სანამ while ციკლს ვისწავლიდეთ, გავიგოთ რა არის "პირობები".

კარელის თავისი სამყაროს შესახებ გარკვეული პირობების გაგება შეუძლია, მაგალითად, შეუძლია გაიგოს მის წინ არის თუ არა კედელი, აქვს თუ არა ბრილიანტები ჩანთაში, დგას თუ არა ბრილიანტზე და ა.შ.

პირობების სიის ნაწილი შემდეგია:

frontIsClear() - არის თუ არა მის წინ სიცარიელე

leftIsClear() - არის თუ არა მისგან მარცხნივ სიცარიელე

rightIsClear() - არის თუ არა მისგან მარჯვნივ სიცარიელე

facingNorth() - იყურება თუ არა ჩრდილოეთისკენ

beepersPresent() - არის თუ არა ბრილიანტი იქ, სადაც თვითონ დგას

წარმოვიდგინოთ, რომ კარელი პირველი გამზირის პირველ ქუჩაზე იმყოფება, ხოლო ბრილიანტი - ბოლო გამზირის პირველ ქუჩაზე და კარელის უნდა ამ ბრილიანტის აღება. კარელმა არ იცის რამხელაა სამყარო და რამდენი გამზირია მასში. გასაგებია, რომ პირველი ქუჩის ბოლოში მისასვლელად move() ბრძანება უნდა გავიმეოროთ რამდენჯერმე. თუმცა პრობლემა იმაშია, რომ არ ვიცით რამდენჯერ. შესაბამისად, for ციკლი ამაში არ გამოგვადგება. თუმცა, აქვეა while ციკლი, რომელიც for ციკლის მსგავსად თავისა და ტანისგან შედგება. while ციკლის ტანში უნდა ჩავწეროთ ბრძანებების მიმდევრობა, რომელთა გამეორებაც გვინდა. while ციკლის თავში კი პირობა, რის შედეგადაც while ციკლი გაიმეორებს ბრძანებების მიმდევრობას **მანამ, სანამ** პირობა სრულდება. მაგალითად შემდეგი კოდის შემთხვევაში

```
while(frontIsClear()){
    move();
}
```

კარელი იმოძრავებს მანამ, სანამ მის წინ სიცარიელეა. შესაბამისად, იმოძრავებს მანამ, სანამ მის წინ კედელი არ იქნება, ანუ მიაღწევს პირველი ქუჩის ბოლოში.

if ოპერატორი

ხანდახან არის მომენტი, როდესაც რაღაც პირობის შემოწმება გჭირდება ხოლმე ერთხელ და არა მრავალჯერ(ანუ მანამ, სანამ), ისე როგორც while ციკლშია. ამ დროს გამოგვადგება if ოპერატორი, რომელიც for-ის და while-ის მსგავსად შედგება თავისა და ტანისაგან. if-ის თავში შეგვიძლია ჩავწეროთ ნებისმიერი პირობა, ხოლო მის ტანში ფუნქციების ნებისმიერი მიმდევრობა, შესაბამისად, **თუ** პირობა სრულდება, მაშინ კარელი შეასრულებს if-ის ტანში ჩამოწერილ ბრძანებებს, თუ პირობა არ სრულდება მაშინ არ შეასრულებს. for-ის და while-ის მსგავსად იმის მისათითებლად თუ რა ფუნქციები წერია if-ის ტანში თითოეული ფუნქცია უნდა მოვაქციოთ ფიგურული ფრჩხილების შიგნით. მაგალითად შემდეგი კოდის შემთხვევაში

```
if(beepersPresent()){
    pickBeeper();
}
move();
```

კარელი შეამოწმებს და თუკი ბრილიანტზე დგას, მაშინ მას აიღებს (if დამთავრდება) და შემდეგ წინ გადაადგილდება, ხოლო თუკი ბრილიანტზე არ დგას, მაშინ if-ის ტანში არსებულ ბრძანებას საერთოდ არ შეასრულებს და უბრალოდ გადაადგილდება.

if ოპერატორს გააჩნია else – "**თუ არა**" ბლოკიც, რომლის საშუალებითაც შეგვიძლია შემდეგი ტიპის ლოგიკა განვავითაროთ - თუკი კარელი ბრილიანტზე დგას, მაშინ აიღოს ის, ხოლო **თუ არა**, მაშინ დადოს ბრილიანტი. კოდი შემდეგნაირად დაიწერება:

```
if(beepersPresent()){
    pickBeeper();
}else{
    putBeeper();
}
```

if ოპერატორს else ბლოკი შეიძლება ჰქონდეს, შეიძლება არა, ანუ, შეგვიძლია დავუწეროთ, შეგვიძლია არ დავუწეროთ. გასათვალისწინებელია, რომ ნებისმიერ შემთხვევაში შესრულდება ან if ბლოკში არსებული ბრძანებები(თუკი პირობა ჭეშმარიტია) ან else ბლოკში არსებული ბრძანებები(თუკი პირობა მცდარია). ამასთან ერთ-ერთი ბლოკი აუცილებლად შესრულდება(რადგან პირობა ან ჭეშმარიტი იქნება, ან მცდარი) და ორივე ვერასდროს შესრულდება.