

## تفسیر برنامه بهینه سازی بوسیله شبیه سازی روزنامه فروش

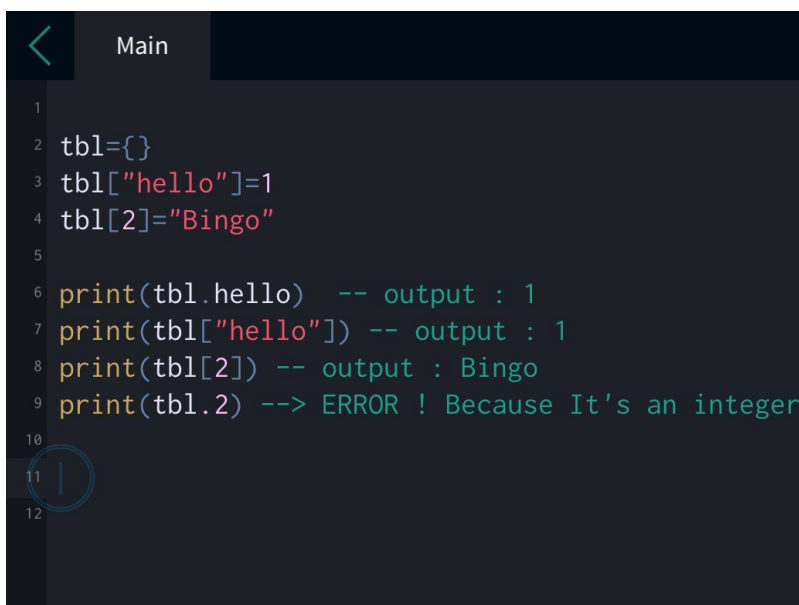
امید یعقوبی

پاییز ۹۴

در ابتدا داده های مسئله را به زبان کد نگاشت کردم. این کار با قابلیت هایی که زبان برنامه نویسی لوآ در اختیارم گذاشته بود بسیار زیبا و به راحتی انجام می شد.

زبان برنامه نویسی لوآ فاقد آبجکت یا کلاس می باشد. اما ابزار به وجود آوردن آن را در اختیار گذاشته است. یکی از این ابزار نوع داده table یا جدول است و دیگری نگاه First-Class Object به توابع که در ادامه هریک را توضیح مختصر خواهم داد.

جدول در زبانهای برنامه نویسی دیگر مانند پایتون و سوئیفت به دیکشنری و در برخی مانند جاوا و سی پلاس پلاس به نگاشت یا Map معروف است. نوعی کلکسیون است که می تواند انواع مختلفی از جفت های کلید-مقدار را در خود نگه دارد. برای مثال نوع داده آرایه را می توان با جدول پیاده سازی نمود، کافیسیت برای کلیدهای آن تناظر یک به یک بین اعداد را از یک تا اندازه ی جدول در نظر گرفت. ولی جدول یا دیکشنری آزادی بیشتری به ما می دهد چون کلیدها می توانند از هر نوعی باشند، رشته یا اعداد حقیقی فرقی نمی کند. قابلیت دیگری که زبان برنامه نویسی لوآ در جدولها برای ما گذاشته است استفاده از عملگر جایگزین کروشه برای دسترسی به مقادیر جدول است، ما می توانیم با کروشه یا نقطه به اعضای که کلیدهایشان از جنس رشته است دسترسی داشته باشیم، در مثال زیر جدولی ساخته و از هر دو عملگر برای دسترسی به مقادیر استفاده کرده ام:



```

1
2 tbl={}
3 tbl["hello"]=1
4 tbl[2]="Bingo"
5
6 print(tbl.hello) -- output : 1
7 print(tbl["hello"]) -- output : 1
8 print(tbl[2]) -- output : Bingo
9 print(tbl.2) --> ERROR ! Because It's an integer
10
11
12

```

همانطور که مشاهده می کنید نمی توانیم به اعضای که کلیدهایشان عددی است با نقطه دسترسی داشته باشیم.

دیگر خصوصیت لوآ که ذکر شد، نگاه First-Class Object آن به توابع است. این به آن معناست که این زبان به توابع نیز مانند اشیاء نگاه می کند و شما می توانید آنها را مانند متغیر به دیگر توابع پاس دهید یا تابعی بنویسید که یک تابع را بعنوان خروجی برمی گرداند، اما دلیل توجه ما به این قابلیت بیشتر از آن روست که می توان آنها را در جداول نیز ذخیره نمود، در مثال صفحه بعد تابعی را به جدول اضافه کرده و از آن استفاده می کنیم.

```

< Main
1
2 tbl={}
3 tbl["property_1"]=120
4 tbl[2]="I Act like a class :)"
5 function tbl:somethingLikeMethod()
6     print (tbl[2])
7 end
8
9 print(tbl.property_1)  -- output : 120
10 print(tbl.somethingLikeMethod()) -- output : I Act like a class :)
11

```

حال ابزار ما برای استفاده از جداول مانند پکیجی به هدف نگه داری خصوصیات و توابع کامل است. هرچند در این زبان قابلیت‌هایی مانند ارث بری و چندریختی وجود ندارد، اما راه حل مناسبیست برای حل مسائلی که پیچیدگی کمتری دارند.

حال بر می گردیم به مسئله خودمان یعنی مسئله روزنامه فروش، مثال سوم از فصل دوم از کتاب شبیه سازی جری بنکس ترجمه محلوچی.

روزنامه فروشی هر روزنامه را به قیمت ۱۳ واحد پولی خریده و هر یک را به ۲۰ واحد پولی می فروشد. در پایان روز روزنامه ها ارزش خود را از دست داده و به عنوان کاغذ باطله به قیمت ۲ واحد پولی برای هرکدام فروخته می شوند.

سه نوع روز وجود دارد، متوسط با احتمال ۴۵ درصد، خوب ۳۵ درصد و بد ۲۰ درصد. احتمال تقاضا برای هر یک از این روزها با احتمالاتی که ذکر آن وقت گیر است و در کتاب و برنامه آمده است پخش شده است. حال می رویم به سراغ برنامه:

```

< Main random_tools tools
1 -- paperSeller simulation
2 -- Written by Omid Yaghoubi
3 -- deopenmail@gmail.com
4 -- Simulation Home Work , fall 2015
5
6 -- page 46
7
8
9
10 function setup()
11     myInit()
12
13     -- initial conditions
14     paper={}
15     paper["price"]={}
16     paper.price["buy"]=13
17     paper.price["sell"]=20
18     paper.price["outOfDate"]=2
19     -- paper man can buy 10 , 20 , 30 , 50 ... papers
20     -- paper man can buy 10*X pack of papers
21     -- point: best amount off papers to buy
22

```

خصوصیات روزنامه را در يك جدول با همان نام ذخیره کرده ام تا از نوشتن اعدادی مانند ۱۳ ، ۲۰ و ۲ پرهیز کرده و خوانایی و وضوح برنامه را بالا ببرم و همچنین با پارامتری کردن آن تغییرات بعدی در برنامه را تسهیل نمودم به گونه ای که برای ورودیهای مختلف به راحتی بتوان آن را تغییر داد.

همانطور که در تصویر نیز مشاهده می شود برنامه دارای سه فایل می باشد، فایل اصلی و دو فایل که در آن توابع کاربردی خود را تعریف نمودم، این دو فایل را در برنامه های مختلف استفاده می کنم به همین علت برخی از توابع آن در این برنامه بلااستفاده است. توضیح برخی از توابعی که در آن پیاده سازی کردم، بعنوان مثال تابع کشیدن محور مختصات و یا تنظیمات اولیه نیز هم خارج از هدف این بحث است و هم ساده و سراسر است، به طوری که توضیح آن را بی فایده فرض کردم.

اما برخی دیگر از ابزار مانند چند تابع در ابزارهای مولد تصادفی با توجه به مبحث پیش رو مهم بوده و به موقع در مورد آن نحوه پیاده سازی آن توضیح خواهم داد.

حال میرسیم به مدل کردن روز و تقاضا. ابتدا جدولی برای روز می سازیم. سپس به آن مشخصه شانس یا احتمال را اضافه می کنیم و برای هر يك از روزهای خوب ، بد و متوسط احتمالات را درج می کنیم. در آخر متدی به جدول اضافه می کنیم که با توجه به توزیع احتمالاتی داده شده يك روز را برگرداند. با توجه به توزیع احتمالاتی یعنی احتمال انتخاب شدن روز متوسط بعنوان روز تصادفی ۴۵ درصد باشد. تابعی که درون این متد دیده می شود از سری توابعی است که هنگام پیاده سازی ابزار مولد تصادفی طراحی کردم و کد آن را در ادامه توضیح خواهم داد.

```
-- ===== day =====
day={}
day["chance"]={}
day["chance"]["good"]=0.35
day["chance"]["med"]=0.45
day["chance"]["bad"]=0.20
function day:rndDay()
    return generateRandomWithChance(day["chance"])
end
-- =====
```

### توضیح کد استفاده شده در تابع (generateRandomWithChance(table[]))

این تابع همانطور که در کد زیر مشاهده می کنید خود از تابع دیگری به نام generateCumulativeChance استفاده می کند:

```
16 function generateRandomWithChance(tbl)
17
18     -- tbl format ==> tbl[name]=chance
19     -- tbl format ==> tbl["apple"]=0.4
20     cumulativeChance=generateCumulativeChance(tbl)
21     rnd=math.random()
22     for k,v in pairs(cumulativeChance)
23     do
24         if (rnd>v.x) and (rnd<=v.y) then
25             return k
26         end
27     end
28 end
```

### کد تابع generateCumulativeChance

**توجه:** منظور از وکتور تک بعدی در نوشته ی زیر دو نقطه در يك فضای تک بعدی است که يك خط جهت دار را تداعی می کنند. مثلاً نقطه ۰ و ۰.۲۵ در يك وکتور تک بعدی نمایانگر خطیست جهت دار که از صفر به سمت ۰.۲۵ پیش می رود و در ۰.۲۵ متوقف می شود.

این تابع احتمال تجمعی را محاسبه کرده و هر احتمال تجمعی را مانند يك وکتور تک بعدی ذخیره می کند. هر وکتور شامل مولفه ی  $x$  و  $y$  می باشد. برای مثال سه احتمال ۲۵ درصد، ۵۵ درصد و ۲۰ درصد را در نظر بگیرید این تابع هر يك از اینها را به يك وکتور تبدیل کرده به طوری اندازه ی هر وکتور همان احتمال آن باشد. دلیل استفاده از وکتور آن است که زبان برنامه نویسی هنگام پیمایش جدول هیچ تضمینی نمی کند که آنها را به ترتیب دلخواه شما محاسبه کند، ممکن است اولین عنصر مورد بررسی دومین عنصری باشد که تعریف نموده اید، از این رو اگر احتمال تجمعی را با آن فرض غلط که ترتیب آن دست نمی خورد در نظر بگیریم برای هر عنصر يك احتمال کافیسست، برای مثال خواهیم داشت ۲۵ و ۷۵ و ۱، ولی در این صورت اگر بگوییم هرکجا که زیر مولفه تابع تجمعی بود یعنی در همان بازه است، با توجه به عدم رعایت ترتیب هنگام پیمایش به مشکل بر خواهیم خورد، فرض کنید عدد رندوم به دست آمده ۰.۰۹ باشد یعنی در بازه ی اولی ولی حلقه ابتدا آخرین خروجی تابع تجمعی یعنی ۱ را در نظر بگیرد، در این صورت شرط درست خواهد بود چون ۰.۰۹ کوچکتر از ۱ است و سومین احتمال را برخواهد گرداند که درست نیست. اما اگر برای هر مولفه يك وکتور داشته باشیم، مستقل از ترتیب بررسی می توانیم مطمئن باشیم که تابع درست کار خواهد کرد. در این صورت مولفه ای که دارای احتمال ۲۵ درصد است ممکن است دارای ایکس و ایگرگ ۰ و ۰.۲۵ باشد یا ممکن است دارای ایکس و ایگرگ ۰.۵۵ و ۰.۸ باشد یا آنکه ایکس آن ۰.۷ و ایگرگ آن ۱ باشد، ولی در هر صورت مطمئن خواهیم بود که اندازه ی آن همیشه ۰.۲۵ است و احتمال انتخاب شدن عنصر متناظر با آن در تابع generateRandomWithChance نیز ۲۵ درصد خواهد بود.

```

1 function generateCumulativeChance(tbl)
2
3     cumulativeChance={}
4     lastChance=0
5     for k,v in pairs(tbl)
6     do
7         cumulativeChance[k]={}
8         cumulativeChance[k]["x"]=lastChance
9         cumulativeChance[k]["y"]=lastChance+tbl[k]
10        lastChance=cumulativeChance[k].y
11    end
12
13    return cumulativeChance
14 end

```

ادامه ی کد مربوط به تابع generateRandomWithChance نیز با توضیحات داده شده کاملن واضح است، این عدد رندمی بین ۰ و ۱ تولید کرده و با پیمایش وکتورهای خروجی generateCumulativeChance بررسی می کند که در رنج کدامیک از آنها قرار دارد و آن را برمی گرداند.

در این صورت با هربار اجرای تابع rndDay به احتمال ۳۵ درصد روز خوب ۴۵ درصد روز متوسط و ۲۰ درصد روز بد داریم.

شاید از من ایراد بگیرید که اگر در هر بار اجرای تابع، ایکس و ایگرگ وکتور تغییر کند ولی نه اندازه ی آن، عددهای تصادفی تولید شده منطبق با توزیع نخواهند بود. اما این طور نیست چرا که فرقی نمی کند که در دایره ای که می چرخد در هر بار چرخش مکان برشها تغییر کند، مهم آن است که آن برشهایی که زاویه ی بیشتری را در برمی گیرند احتمال بیشتری برای انتخاب شدن دارند و این زاویه هاست که نباید تغییر کند نه مکان برشها. در اینجا زاویه همان اندازه وکتورهاست که بدون تغییر باقی می ماند.

برای آنکه گمانه زنی بدون آزمایش نکرده باشم، من کدی نوشتم که صحت کارکرد توابعم را نشان می دهد. کد را در تصویر زیر آورده ام و به حدی گویاست که نیازی به توضیح ندارد، تنها نکته مهم آن است که بدانید تابع draw در هر ثانیه حدود ۶۰ بار اجرا می شود:

```
6   tbl={}
7   tbl["chance"]={}
8   tbl.chance[25]=0.25
9   tbl.chance[43]=0.43
10  tbl.chance[32]=0.32
11  counter={}
12  counter[25]=0
13  counter[43]=0
14  counter[32]=0
15  sum=0
```

```
25 function draw()
26   sum=sum+1
27   rnd=generateRandomWithChance(tbl.chance)
28   counter[rnd]=counter[rnd]+1
29 end
```

و خروجی برنامه بعد از ۱۰۰۰۰ بار اجرای تابع draw :

```
Parameters <
round(counter[25]/sum)
25.62
round(counter[43]/sum)
42.38
round(counter[32]/sum)
31.98
sum
10204
```

مشاهده می کنید که اعداد تصادفی مطابق توزیعشان پخش شده اند.

```

34  -- ===== demand =====
35  demand={}
36  demand["good"]={}
37  demand["med"]={}
38  demand["bad"]={}
39
40  demand.good["chance"]={}
41  demand.med["chance"]={}
42  demand.bad["chance"]={}
43
44  demand.good.chance[40]=0.03
45  demand.good.chance[50]=0.05
46  demand.good.chance[60]=0.15
47  demand.good.chance[70]=0.20
48  demand.good.chance[80]=0.35
49  demand.good.chance[90]=0.15
50  demand.good.chance[100]=0.07
51
52  demand.med.chance[40]=0.1
53  demand.med.chance[50]=0.18
54  demand.med.chance[60]=0.4
55  demand.med.chance[70]=0.2
56  demand.med.chance[80]=0.08
57  demand.med.chance[90]=0.04
58  demand.med.chance[100]=0.0
59
60  demand.bad.chance[40]=0.44
61  demand.bad.chance[50]=0.22
62  demand.bad.chance[60]=0.16
63  demand.bad.chance[70]=0.12
64  demand.bad.chance[80]=0.06
65  demand.bad.chance[90]=0.00
66  demand.bad.chance[100]=0.00

```

برگردیم به مسئله روزنامه فروش و مدلسازی تقاضا. توزیع احتمالی تقاضا با توجه به روزهای خوب، بد و متوسط متفاوت است، ابتدا جدولی ساخته ام که نمایانگر تقاضاست، به آن سه فیلد روز خوب، بد و متوسط اضافه کرده ام و برای هرکدام توزیع گفته شده در کتاب را لحاظ کرده ام.

سپس به هرکدام از روزهای خوب، بد و متوسط تقاضا یک تابع انتخاب رندوم بر اساس جدول منتسب کرده ام.

```

68  function demand.good.rndDemand()
69      return generateRandomWithChance(demand.good["chance"])
70  end
71
72  function demand.med.rndDemand()
73      return generateRandomWithChance(demand.med["chance"])
74  end
75
76  function demand.bad.rndDemand()
77      return generateRandomWithChance(demand.bad["chance"])
78  end
79

```

دلیل انتساب سه تابع به جای یک تابع را در ادامه متوجه خواهید شد. حال تنها برای آنکه کنجکاویتان برطرف شود، نگاهی به کد زیر بیندازید.

```

117  currentDay=day.rndDay()
118  currentDemand=demand[currentDay].rndDemand()

```

سپس عملیاتی که می توان انجام داد را به صورت تابع مدل کرده ام. عملها شامل فروش روزنامه، خرید روزنامه و فروش روزنامه های باطله می باشد.

```

172  function buyPaper(amount)
173      newDeposit=amount
174      return newDeposit, (paper.price.buy*amount)
175  end
176
177  function sellPaper(amount,yourDeposit)
178      result=math.min(amount,yourDeposit)
179      newDeposit=yourDeposit-amount -- could be a negative number
180      return newDeposit, (result*paper.price.sell)
181  end
182
183  function sellOutOfDatePaper(amount)
184      return amount*paper.price.outOfDate
185  end

```

تابع اولی مقدار مورد نظر برای خرید را می گیرد و دو مقدار برمی گرداند. اولی موجودی جدید روزنامه که فرض بر آن است که قبل از هر خرید صفر است (می توان آن را تغییر داد) دومی نیز میزان پولی که از دست رفته است.

تابع فروش ، دو مقدار برمی گرداند، اولی موجوی جدید روزنامه هاست و دومی پولیست که بخاطر فروش به دست آمده است. از آنجا که ممکن است میزان تقاضا بیشتر از موجوی باشد تابع بین تعداد موجودی روزنامه شما و تعداد تقاضا مینیموم میگیرد. همچنین کنار موجودی جدید نوشته ام که می تواند عددی منفی باشد که نشانگر بالاتر بودن میزان تقاضاست و به وسیله ی آن عدد منفی ما می توانیم سود از دست رفته بخاطر فرونی تقاضا را نیز محاسبه کنیم.

رفتار تابع فروش روزنامه های باطله نیز واضح است.

حال وارد قسمت تکرار شونده ی برنامه می شویم. داخل حلقه در هر قدم يك واحد (یا ده واحد با توجه به داده های مسئله که گفته است روزنامه ها در بسته های ۱۰ تایی خریداری می شوند.) به تعداد بسته های خریدای شده اضافه می شود، سپس در حلقه ای دیگر جمع سود در يك سال (یا هر تعداد روزی که شما تنظیم کنید) محاسبه می شود. بیرون حلقه ی داخلی میانگین سود در هرروز محاسبه می شود. سپس این داده ها را در جدولی ذخیره می کنیم که کلیدهای آن تعداد بسته های خریداری شده و مقادیر آن میانگین سود روزانه است. توسط این جدول نموداری می کشیم و ماکسیمم را که تعداد خرید بهینه است نشان می دهیم. در ادامه به توضیح مختصر کد گفته شده خواهیم پرداخت.

لازم به ذکر است که تمامی کدهایی که گفته می شود در تابع draw نوشته شده و این به این معنیست که آنها به صورت مداوم اجرا خواهند شد (البته شرط پایانی نیز لحاظ کرده ام تا پس از تکمیل نمودار نمودار جدیدی ترسیم نکند). من با استفاده از خاصیت تکرار شونده ی این تابع حلقه بیرونی را حذف کردم، با اینکار هنگام اجرای برنامه می توان شاهد حرکت نقاط در نمودار برای رسیدن به نقطه بهینه به صورت انیمیشن بود که زیبایی خروجی را دوچندان کرده است.

#### کد مربوط به حلقه درونی:

```

117 sumProfit=0
118 for dayCounter=1,countOfDays do
119
120     currentDay=day.rndDay()
121     currentDemand=demand[currentDay].rndDemand()
122     deposit,moneyLossForBuying=buyPaper(buyAmount)
123     deposit,moneyEarnFromSelling=sellPaper(currentDemand,deposit)
124     moneyEarnFromOutOfDates=0
125     profitLoss=0
126     if deposit>0 then
127         moneyEarnFromOutOfDates=sellOutOfDatePaper(deposit)
128         --print(moneyEarnFromOutOfDates.." ",",deposit) -- for debug porpus
129     else
130         profitLoss=math.abs(deposit)*( paper.price.sell - paper.price.buy)
131         --print(profitLoss) -- for debug porpus
132     end --end if
133
134     profit=moneyEarnFromSelling-moneyLossForBuying-profitLoss+moneyEarnFromOutOfDates
135     sumProfit=sumProfit+profit
136
137 end -- end loop ( from day 1 to countOfDays for example 1 year )

```

همانطور که مشاهده می کنید، ابتدا روزی را به تصادف و با توجه به توزیع احتمالی آن انتخاب می کند، سپس با توجه به روز تصادفی تقاضای تصادفی را به دست می آورد. سپس با توجه به آنکه buyAmount در هر قدم حلقه بیرونی تغییر می کند، يك تعداد روزنامه می خرد و هزینه از دست رفته بخاطر خرید و موجودی روزنامه را به روز می کند، سپس با توجه به تقاضا روزنامه ها را می فروشد و همزمان هزینه به دست آمده از فروش و موجودی روزنامه را آپدیت می کند. در اینجا سه

حالت احتمال دارد که به وجود بیاید، یا تمام روزنامه ها به فروش رفته اند و تقاضا دقیقا برابر موجودی بوده، در این حالت وارد else شده اما از آنجا که موجودی صفر است کل عبارت درون else صفر شده و تغییری نمی کند. حالت دوم اینکه موجودی روزنامه از تقاضا بیشتر باشد، در این صورت deposit بیشتر از صفر بوده و روزنامه های مانده بعنوان کاغذباطله فروخته می شوند. حالت سوم زمانیست که تقاضا بیشتر از موجودی باشد که در این صورت سود از دست رفته محاسبه می شود.

در نهایت سود نهایی در روز محاسبه شده و به جمع سودها برای میانگین گیری (بعد خروج از حلقه) اضافه می شود.

کد مربوطه برای میانگین گیری و ذخیره آن در جدول برای کشیدن نمودار:

```

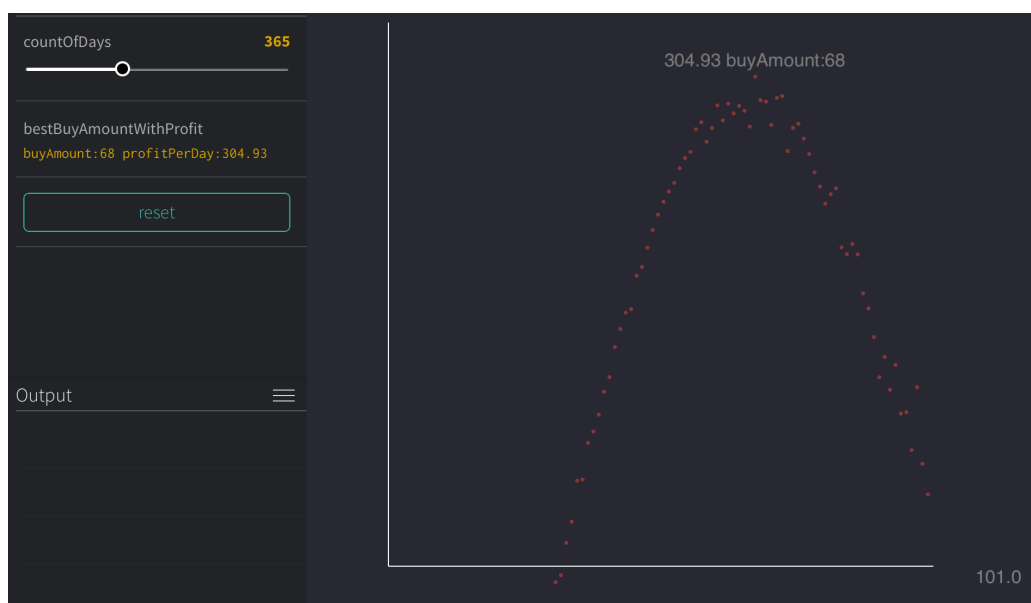
138 -- calculate mean:
139 profitPerDay=sumProfit/countOfDays
140
141 if profitPerDay>maxProfitPerDay then
142     maxProfitPerDay=profitPerDay
143     bestBuyAmountWithProfit="buyAmount:.."buyAmount.." profitPerDay:.."round(profitPerDay)
144 end -- end if profit ...
145
146 profitPerDayList[buyAmount]=profitPerDay
147 end -- end if buy amount reach to max

```

نکته خاصی در مورد کد بالا قابل ذکر نیست جز اینکه خط ۱۴۳ هدف نمایشی دارد و کاری به محاسبات نخواهد داشت.

ادامه ی کد، نمایش نمودار و نرمالایز کردن اعضای جدول است که خارج از هدف این نوشتار است. در ادامه چند خروجی از برنامه را بررسی می کنیم.

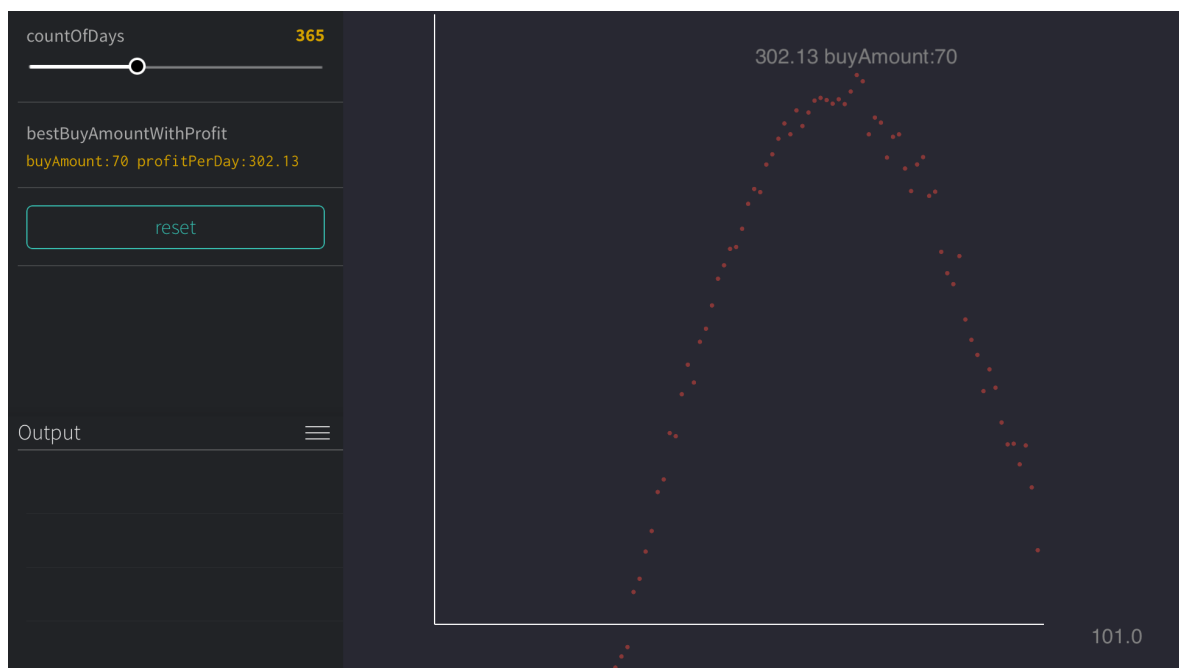
خروجی اول برای ۳۶۵ روز با قدمهای ۱ واحدی



جواب: ۶۸ واحد خرید با بهترین سود میانگین ۳۰۴.۹۴ واحد پولی در روز

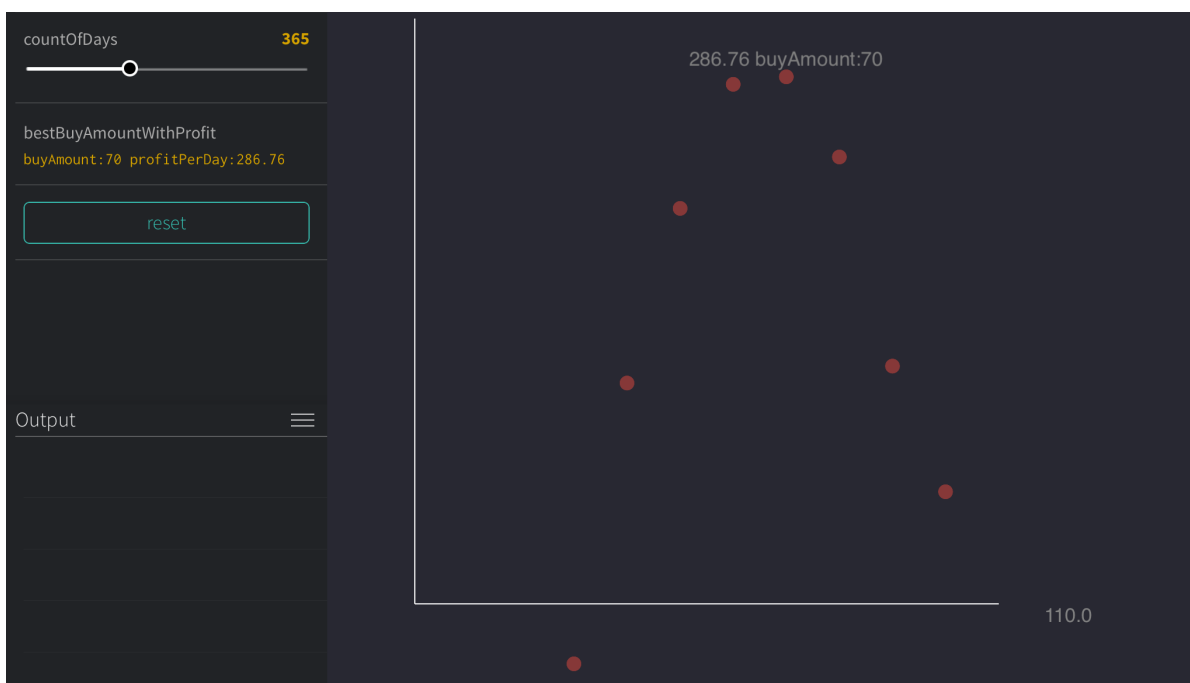


## خروجی دوم برای ۳۶۵ روز با قدمهای ۱ واحدی



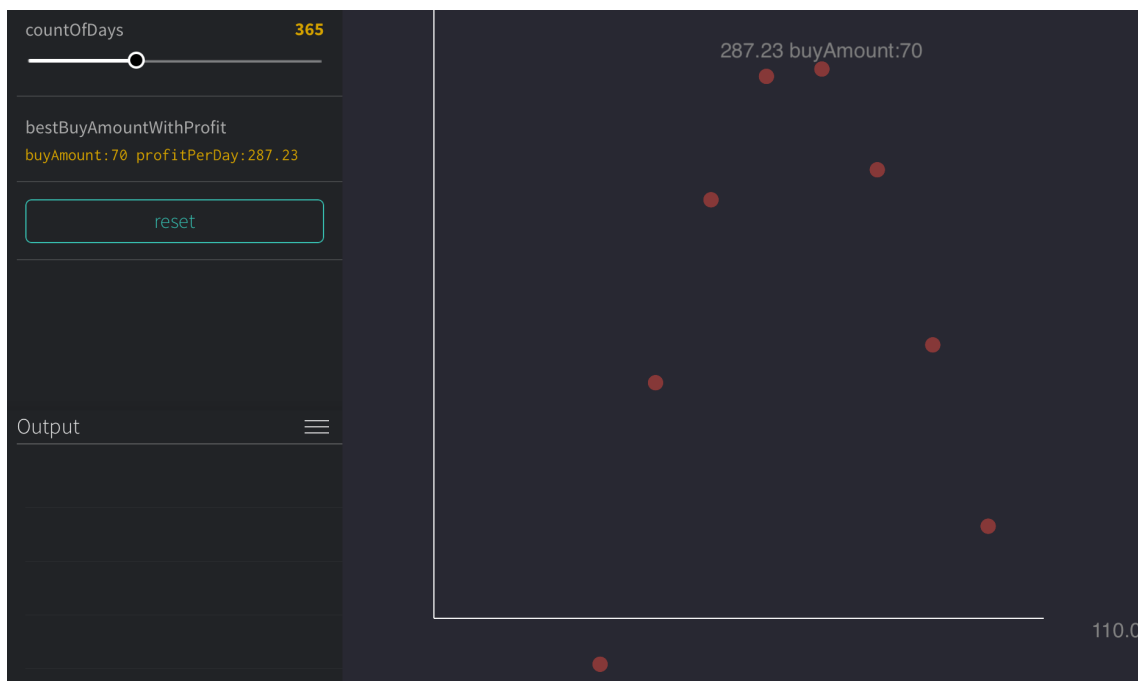
جواب: ۷۰ واحد خرید با بهترین سود میانگین ۳۰۲.۱۳ واحد پولی در روز

## خروجی اول برای ۳۶۵ روز با قدمهای ۱۰ واحدی:



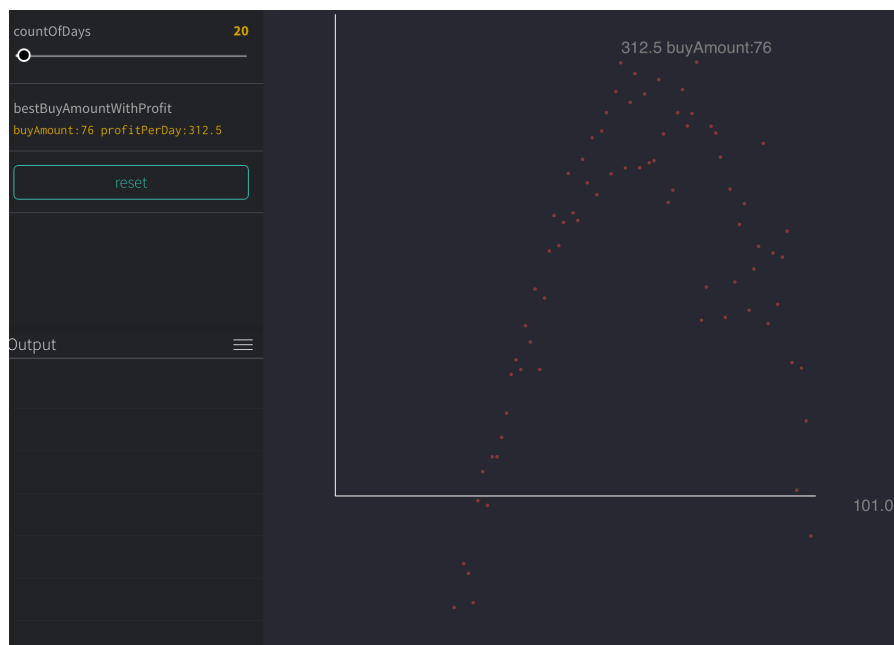
جواب: ۷۰ واحد خرید با بهترین سود میانگین ۲۸۶.۷۶ واحد پولی در روز

خروجی دوم برای ۳۶۵ روز با قدمهای ۱۰ واحدی:



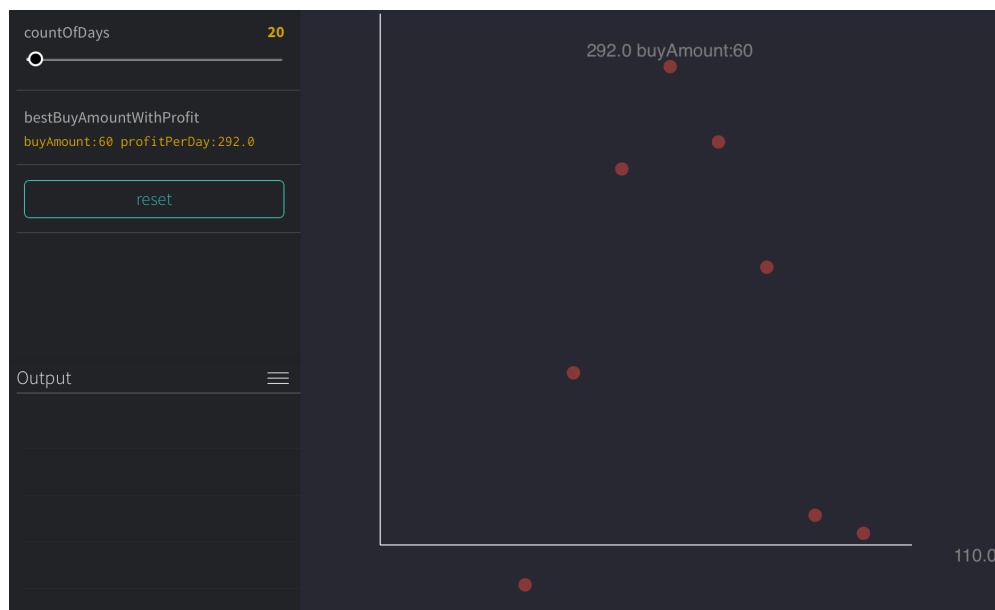
جواب: ۷۰ واحد خرید با بهترین سود میانگین ۲۸۷.۲۳ واحد پولی در روز

خروجی برای ۲۰ روز با قدمهای ۱ واحدی:



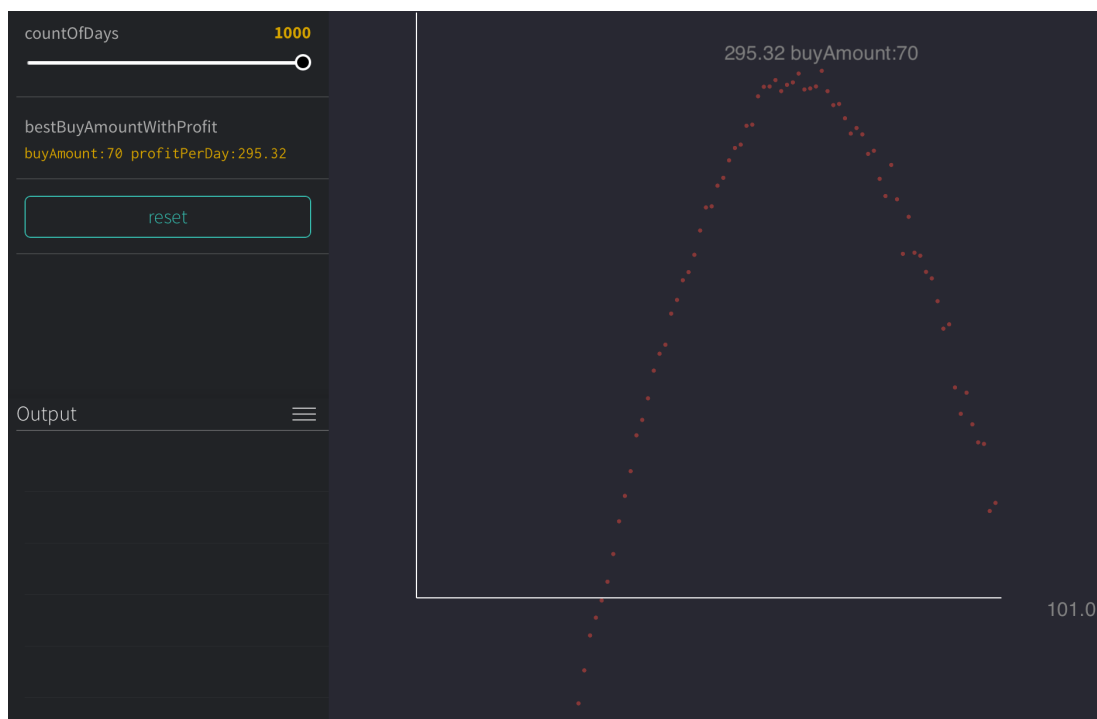
جواب: ۷۶ واحد خرید با بهترین سود میانگین ۳۱۲.۵ واحد پولی در روز

خروجی برای ۲۰ روز با قدمهای ۱۰ واحدی:



جواب: ۶۰ واحد خرید با بهترین سود میانگین ۲۹۲ واحد پولی در روز

خروجی برای ۱۰۰۰ روز با قدمهای ۱ واحدی:



جواب: ۷۰ واحد خرید با بهترین سود میانگین ۲۹۵.۳۲ واحد پولی در روز

با توجه به خروجیها می توان نتیجه گرفت که میزان خرید بهینه ۷۰ واحد در روز می باشد.

- تهیه و تنظیم: امید یعقوبی
- پست الکترونیک [deopenmail@gmail.com](mailto:deopenmail@gmail.com)