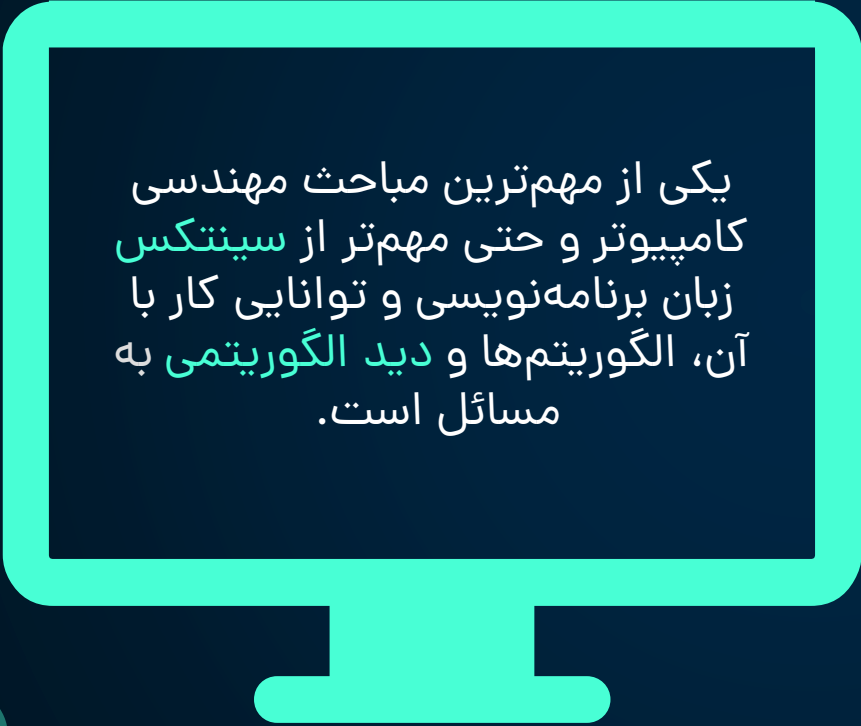




بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

شبهه کد

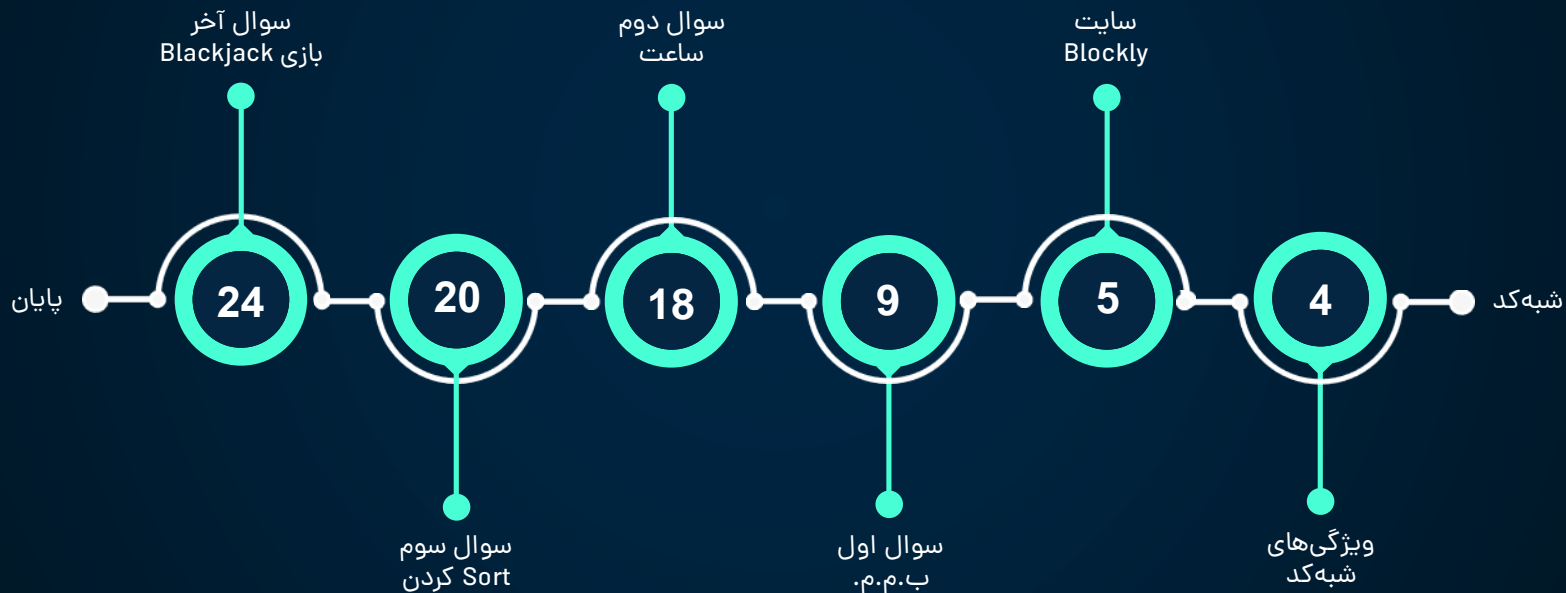
جلسه دوم



یکی از مهم‌ترین مباحث مهندسی
کامپیوتر و حتی مهم‌تر از سینتکس
زبان برنامه‌نویسی و توانایی کار با
آن، الگوریتم‌ها و دید الگوریتمی به
مسائل است.

برای تمرین این توانایی، در طی این
جلسه بدون استفاده از زبان برنامه‌نویسی
و با استفاده از شبه‌کدها سعی به حل
سوالات با تکیه بر الگوریتم داریم.

فهرست



ویژگی‌های شبه کد

- ✓ کمک به پیاده‌سازی راحت‌تر الگوریتم‌های مورد نیاز
- ✓ عدم نیاز به تسلط بر زبان‌های برنامه‌نویسی
- ✓ غیر قابل اجرا در کامپیوتر
- ✓ قابل فهم برای انسان‌ها به دلیل نزدیکی آن به زبان انسان
- ✓ ساده‌تر کردن حل مسئله با ایجاد دید بهتر نسبت به نیازهای سوال

سایت Blockly

همان‌طور که گفته شد، شبه‌کدها به صورت نوشته‌های متنی نوشته می‌شوند که قابل اجرا نیستند.

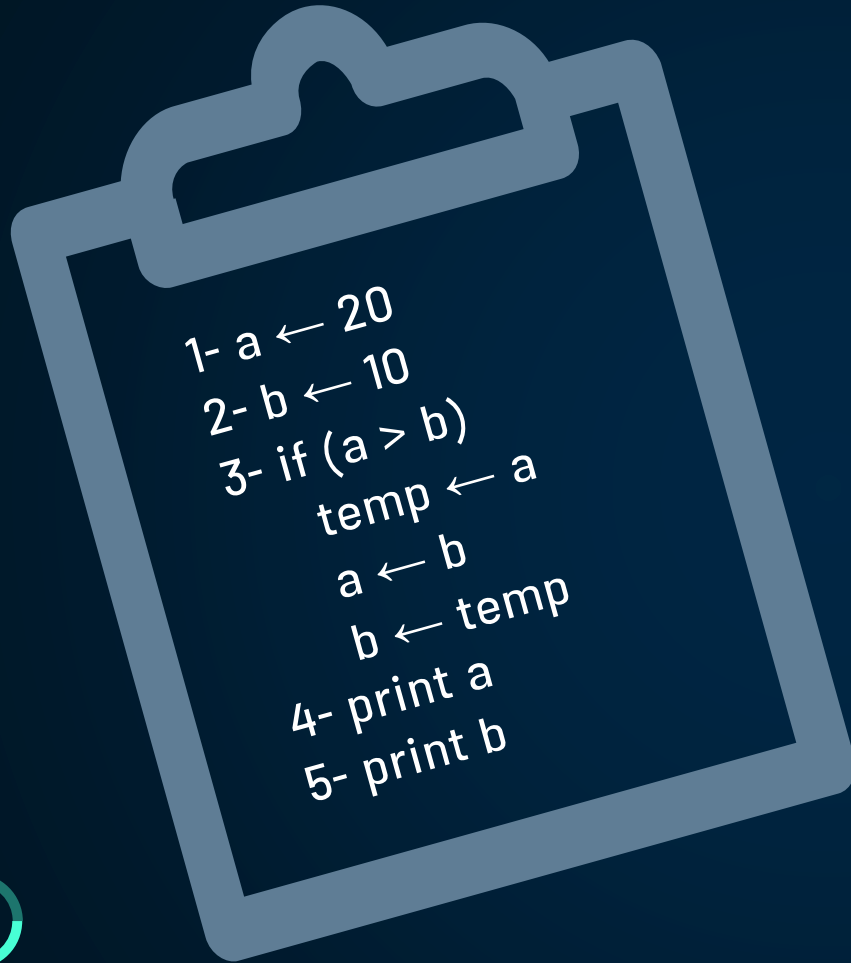
اما برای نزدیک‌تر شدن به مفهوم کد ما از سایتی کمک می‌گیریم که شبه‌کد را به برنامه تبدیل می‌کند تا قابل اجرا باشد و خروجی آن مشاهده شود.

(در ابتدا با فیلترشکن وارد شوید، سپس می‌توانید فیلترشکن خود را خاموش کنید)



Blockly

<https://b2n.ir/427876>



برای تمرین و آشنایی با سایت سعی 

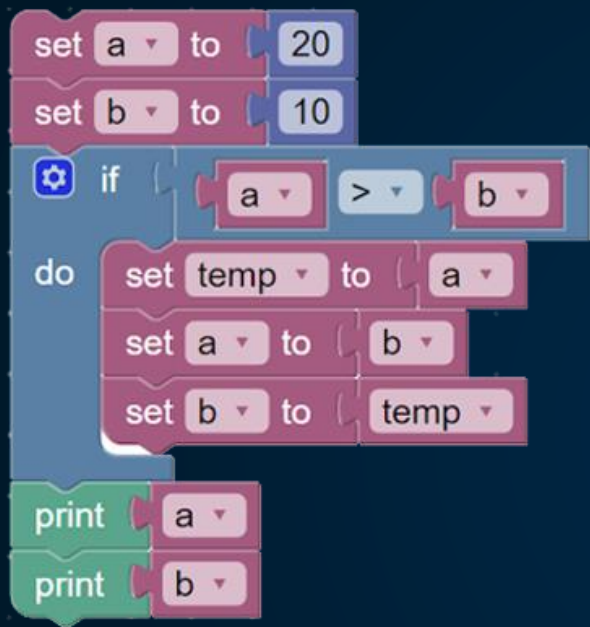
کنید شبه‌کد روبه‌رو را در سایت

Blockly طراحی کنید.

شبه‌کد جابه‌جا کردن دو عدد در

صورت بزرگ‌تر بودن عدد اول:

خروجی حاصل باید چنین شکلی باشد:



حال با زدن روی دکمه‌ی play یعنی ▶ در گوشه‌ی بالا سمت راست صفحه می‌توانید خروجی شبه‌کد خود را در دو مرحله مشاهده کنید.

blockly-demo.appspot.com says

20

OK

blockly-demo.appspot.com says

10

OK

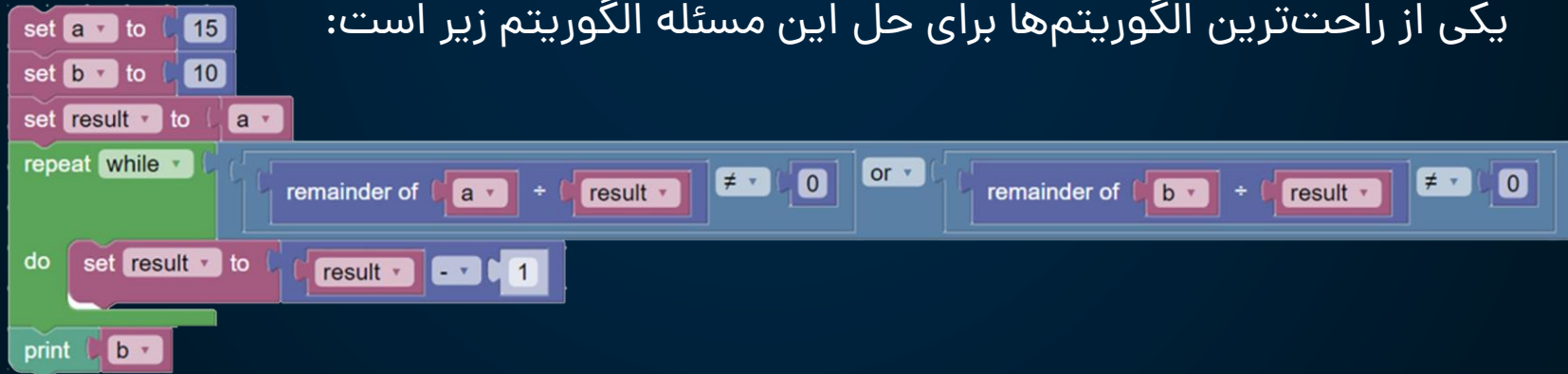
سوال اول: ب.م.م.

فرض کنید برای حل مسئله‌ای نیاز به پیدا کردن بزرگ‌ترین مخرج مشترک دو عدد دارید.

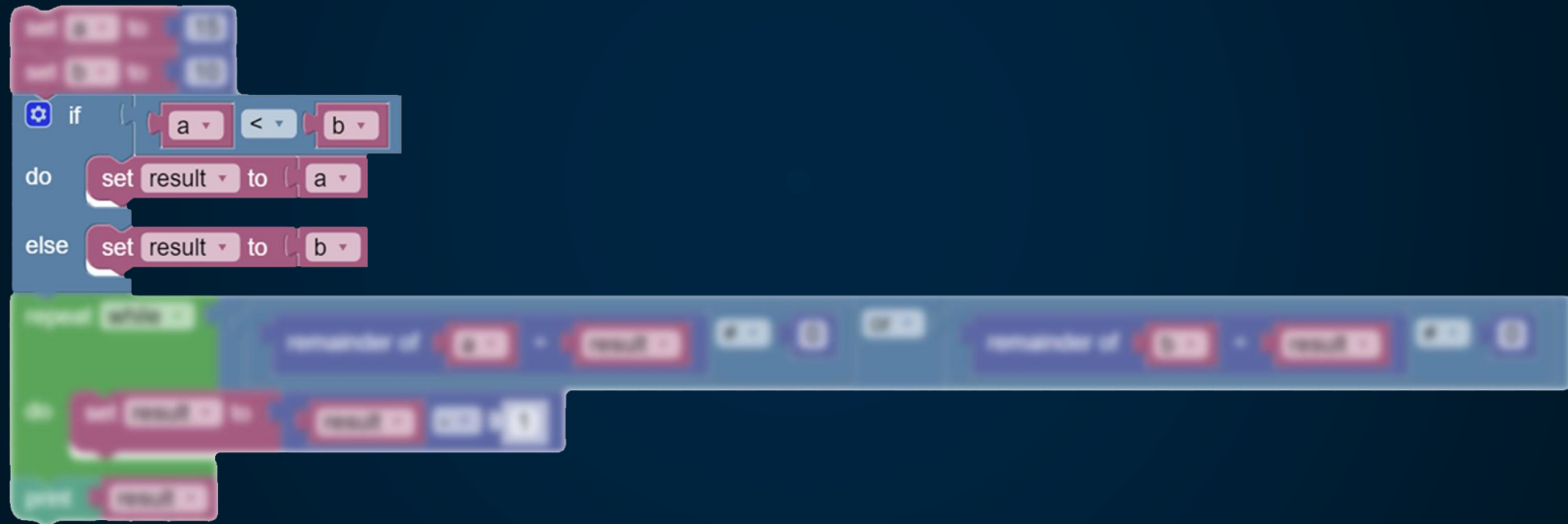
شبه‌کدی بنویسید که بتواند این کار را انجام دهد و با داشتن ۲ عدد، ب.م.م. آن‌ها را حساب کند و خروجی دهد.

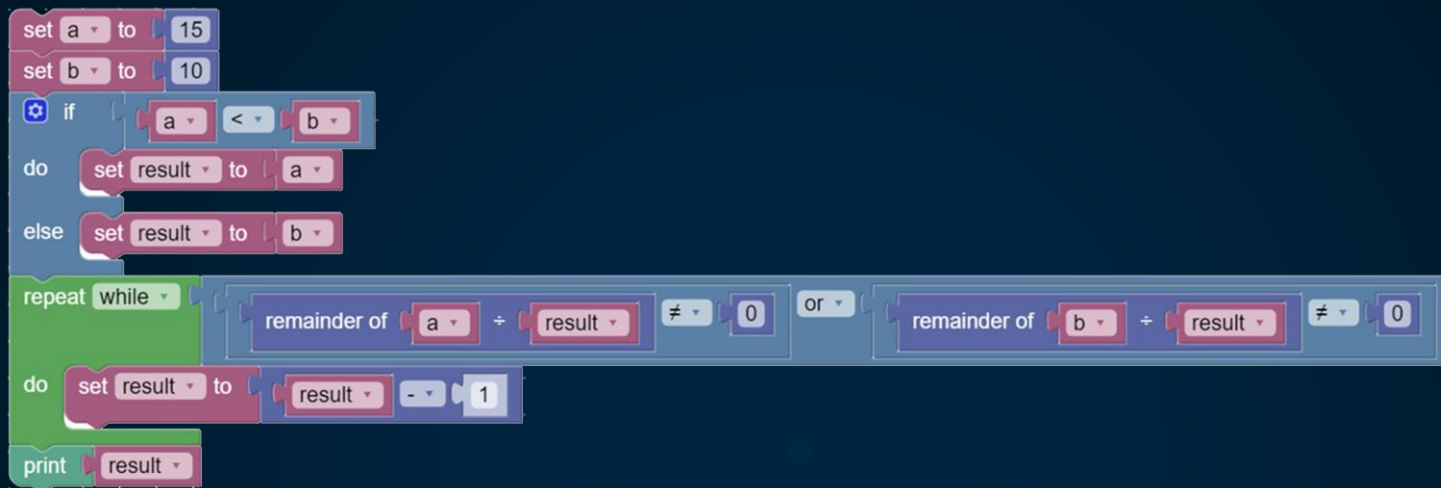
✓ برای حل اکثر مسائل بیش از یک راه حل وجود دارد و با تغییر طرز تفکر و در اصل تغییر الگوریتم مورد استفاده می توان به طریق دیگری مسئله را مدل سازی و حل کرد به شکلی که از نظر زمانی سریع تر به جواب برسیم یا برای رسیدن به جواب حافظه ی کمتری اشغال کنیم.

✓ البته در درس مبانی کامپیوتر هدف یافتن الگوریتم بهینه نیست، اما گاهی سعی می کنیم در صورت امکان برای یک مسئله راه های مختلفی را بیابیم. یکی از راحت ترین الگوریتم ها برای حل این مسئله الگوریتم زیر است:



به نظر شما این شرط چه تغییری در الگوریتم ایجاد می‌کند؟  

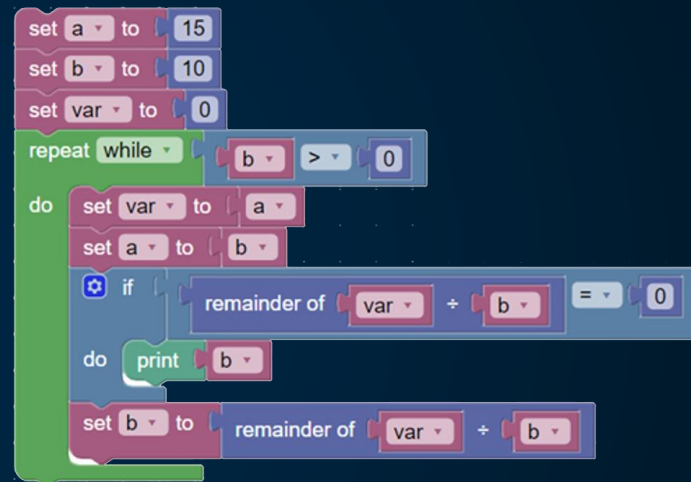
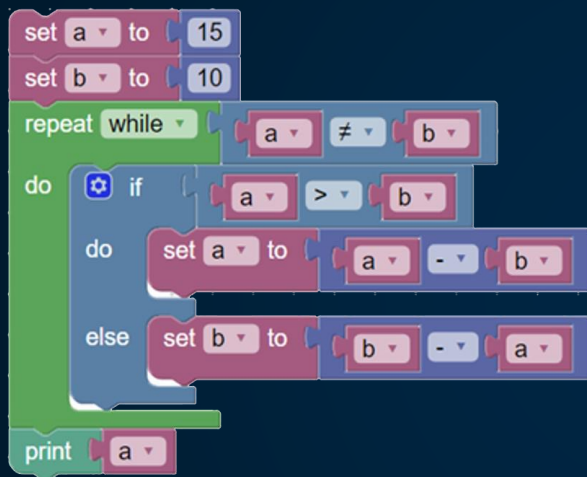




گاهی اوقات اضافه کردن برخی شرطها به الگوریتم باعث بهینه‌تر شدن آن می‌شود. اما گاهی اوقات لازم است که خود الگوریتم تغییر پیدا کند تا پاسخ مسئله در زمان کمتر و یا با استفاده‌ی کمتری از حافظه پیدا شود.

چه الگوریتم دیگری برای ب.م.م.م. پیشنهاد می‌دهید؟

در تصویر زیر الگوریتم دیگری به نام **Euclidean Algorithm** در دو شیوه برای پیدا کردن ب.م.م. نوشته شده است.



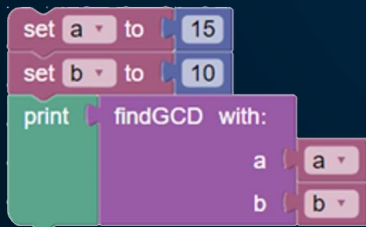
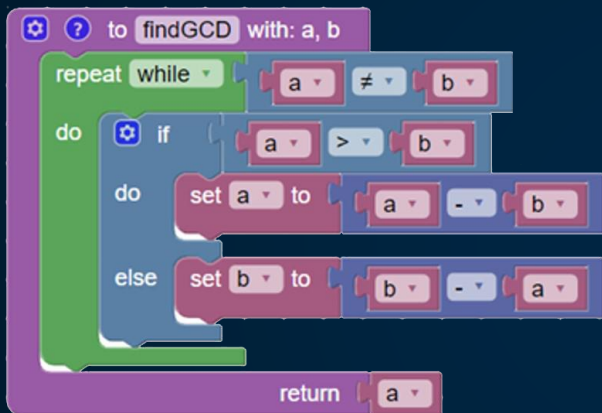
به نظر شما تفاوت این ۲ شبه کد چیست؟



آنها را با هم مقایسه کنید و مزایا و معایب هر کدام را بررسی کنید. به نظر شما هر کدام در چه شرایطی عملکرد بهتری دارند؟

در صورتی که الگوریتم‌های بالا را به صورت تابع‌های جداگانه بنویسیم بهتر می‌توانیم عملکرد آن دو را مقایسه کنیم.

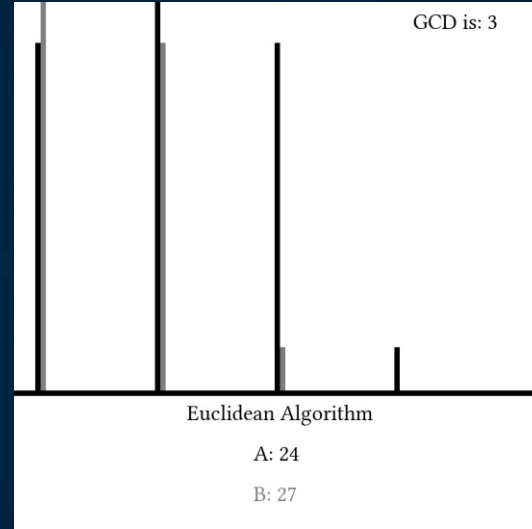
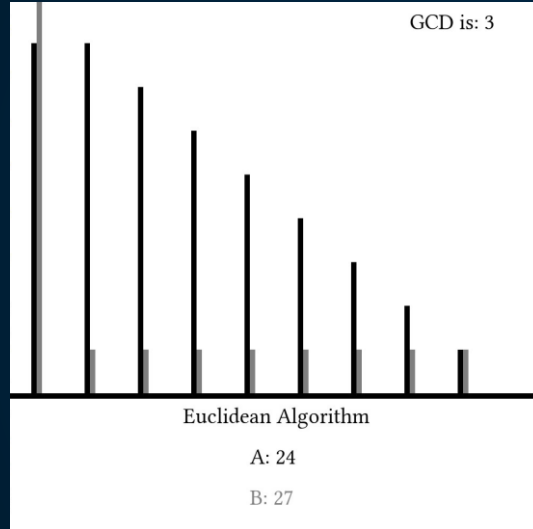
در شکل یکی از الگوریتم‌ها به صورت تابع نوشته شده است. سعی کنید با هم‌گروهی خود الگوریتم دیگر را به صورت تابع بنویسید.



مراحل اجرای دو الگوریتم را برای دو عدد ۲۴ و ۲۷ در نمودارهای زیر بررسی کنید. 

```
to findGCD with: a, b
  repeat while
    do if
      do set a to a - b
      else set b to b - a
    return a
```

```
set a to 15
set b to 10
print findGCD with:
  a
  b
```



به نظر شما آیا این مسئله راه حل دیگری دارد؟ 

برای درک بهتر این الگوریتمها و تفاوت میان آنها می‌توانید از لینک‌های زیر کمک بگیرید.



The Euclidean Algorithm

<https://b2n.ir/430135>



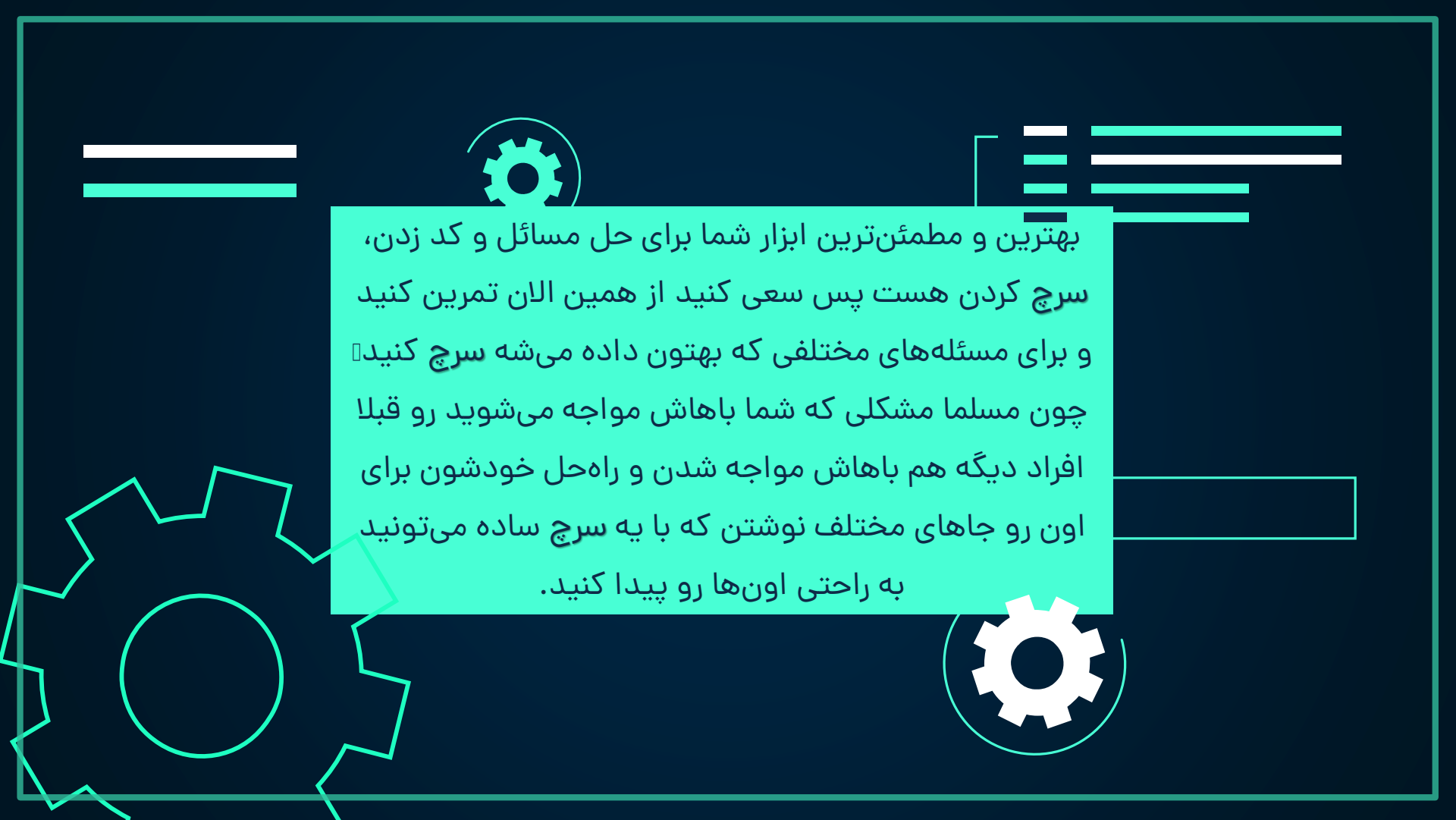
The Euclidean Algorithm

<https://b2n.ir/540981>



Euclidean Algorithm Film

<https://b2n.ir/186162>

The background is a dark blue gradient. It features several decorative elements: a large white gear outline in the bottom left corner, a smaller white gear outline in the top center, and another white gear outline in the bottom right corner. There are also several horizontal white lines of varying lengths and thicknesses scattered across the top and right sides of the image.

بهترین و مطمئن‌ترین ابزار شما برای حل مسائل و کد زدن،
سرچ کردن هست پس سعی کنید از همین الان تمرین کنید
و برای مسئله‌های مختلفی که بهتون داده می‌شه سرچ کنید
چون مسلماً مشکلی که شما باهاش مواجه می‌شوید رو قبلاً
افراد دیگه هم باهاش مواجه شدن و راه‌حل خودشون برای
اون رو جاهای مختلف نوشتن که با یه سرچ ساده می‌تونید
به راحتی اون‌ها رو پیدا کنید.

سوال دوم: ساعت

حلقه‌ها در برنامه‌نویسی به چه علت استفاده می‌شوند؟

چه مفاهیم و اتفاقاتی در دنیای اطراف همواره در حال تکرار شدن هستند؟

ساعت را می‌توان به عنوان یکی از وسایلی دانست که دنیای آن به حلقه یا loop محدود شده است. به نظر شما آیا یک حلقه برای در دست گرفتن زمان کافی می‌باشد؟



اگر یک حلقه داخل حلقه‌ای دیگر استفاده شود به آن **حلقه‌ی تو در تو** یا **nested loop** گفته می‌شود. در این نوع حلقه‌ها به ازای اجرای هر بار حلقه بیرونی، حلقه داخلی به‌طور کامل انجام می‌شود.



ساعت نمونه‌ی خوبی است که مانند حلقه‌های تو در تو عمل می‌کند. به این صورت که برای یک حرکت عقربه ساعت‌شمار به عدد بعدی لازم است تا عقربه‌ی دقیقه‌شمار یک دور کامل بچرخد.



حال از شما می‌خواهیم که در گروه خود با توجه به توضیحات بالا شبه‌کدی برای شبیه‌سازی یک ساعت دیجیتال بنویسید که ساعت، دقیقه و ثانیه را برای یک شبانه روز کامل چاپ کند.



آیا می‌توانید تعداد دفعات اجرای درونی‌ترین حلقه را محاسبه کنید؟

سوال سوم: sort کردن

فرض کنید برای حل مسئله‌ای نیاز به مرتب کردن تعدادی از اعداد داریم.

به عنوان مثال اگر بخواهیم این رشته از اعداد را مرتب کنیم، ذهن ما برحسب عادت این کار را به چه صورت انجام می‌دهد؟ $\{3, 11, 5, 2, 17, 4, 1\}$
راه‌های مختلفی هست. حال باید طوری عمل کنیم که این روند را با کمک الگوریتم‌ها برای کامپیوتر نیز قابل درک کنیم.

اولین الگوریتمی که می‌خواهیم بررسی کنیم، الگوریتم مرتب‌سازی حبابی یا bubble sort است.

برای اینکه بهتر متوجه شوید که این الگوریتم چطور عمل می‌کند می‌توانید از طریق لینک زیر مراحل مرتب‌سازی را به ازای مجموعه اعداد دلخواه مشاهده کنید.



Bubble Sort Algorithm Visualizer

<https://b2n.ir/218225>

حال سعی کنید با هم‌گروهی خود شبه‌کدی برای این الگوریتم بنویسید.



الگوریتم دیگری برای مرتب‌سازی که می‌خواهیم با آن آشنا شویم **selection sort** است.

این بار نیز برای متوجه شدن روند این الگوریتم که دقیقا به چه شکل این مرتب‌سازی را انجام می‌دهد به لینک‌های زیر مراجعه می‌کنیم.



Selection Sort Algorithm

<https://b2n.ir/600734>



Selection Sort Algorithm Visualizer

<https://b2n.ir/489622>

👤👤 در زیر روند الگوریتم را به صورت کلی مشاهده می‌کنید. سعی کنید با هم‌گروهی خود
شبه‌کد این الگوریتم را به گونه‌ای بنویسید که توسط کامپیوتر قابل درک باشد.

- 1- Find the smallest card. Swap it with the first card.
- 2- Find the second-smallest card. Swap it with the second card.
- 3- Find the third-smallest card. Swap it with the third card.
- 4- Repeat finding the next-smallest card, and swapping it into the correct position until the array is sorted.

الگوریتم‌های مرتب‌سازی خیلی زیادن و فقط محدود به
این دو مورد نیستن که می‌تونیم با یه سرچ کوچولو
پیداشون کنیم :

❓ و اما سوال آخر: بازی Blackjack

اگه یادتون باشه گفتیم که دو نفر از برترین برنامه‌نویس‌های دنیا (گُدخدا و Botfather) قصد دارن که به دنیا ثابت کنن هرچیزی رو در دنیا می‌شه به کد تبدیل کرد و برای دستیابی به این هدف لازمه که یه تیم قوی از برنامه‌نویس‌ها و مهندسين کامپیوتر رو جمع‌آوری کنن. برای همین بخشی از دستورکار کارگاه‌ها رو در اختیار گرفتن تا بتونن افراد بیشتری رو به گروه خودشون جذب کنن و در روند آموزش تحولی ایجاد کنن.

به دلیل اهمیت بالای شبه‌کد و توانایی درک الگوریتم، آن‌ها این جلسه از کارگاه‌های مبانی را به عنوان شروع کار خود انتخاب کرده‌اند. تا انتهای این جلسه می‌خواهیم روی الگوریتم بازی Blackjack تمرکز کنیم و شبه‌کد گُدخدا و Botfather را تکمیل کنیم تا با هدف و کار آن‌ها بیشتر آشنا شویم.



بازی Blackjack یک بازی با اعداد ۱ تا ۱۰ است که هر کدام روی تعدادی کارت نوشته شده‌اند. در یک طرف بازی، کارت پخش‌کن (Dealer) و در طرف دیگر بازیکن است. در ابتدای هر دور به صورت تصادفی دو کارت به بازیکن و دو کارت به Dealer داده می‌شود که بازیکن تا پایان بازی تنها یک کارت Dealer را می‌بیند. پس از آن، بازیکن دو گزینه برای بازی کردن دارد (stand / hit) که در ادامه به توضیح آن‌ها خواهیم پرداخت.



هدف بازی، رساندن مجموع ارزش کارت‌های هر دست به عدد ۲۱ بدون رد کردن آن است و هر کدام از دو طرف بازی که بتواند به ۲۱ برسد برنده است و دستی که بیش‌تر از ۲۱ شود دست سوخته می‌باشد. در صورتی که هیچ کدام از شرایط فوق اتفاق نیفتد اگر مجموع ارزش دست‌ها برابر باشد بازی مساوی است و در غیر این صورت دستی که به ۲۱ نزدیک‌تر است برنده بازی می‌شود.



تصمیماتی که بازیکن می‌تواند در بازی بگیرد به صورت زیر است:

Hit: درخواست یک کارت جدید می‌کند و یک کارت تصادفی به دستش اضافه می‌شود.

Stand: کارت جدیدی نمی‌خواهد و دست خود را می‌بندد پس از آن به دست Dealer تا سقف مجموع ۱۷، کارت اضافه می‌شود و مقایسه بین دو دست صورت می‌گیرد تا نتیجه بازی مشخص شود و بازی به پایان می‌رسد.

```
set dealer_hand to RANDOM(1,10)
```

```
set your_hand to RANDOM(1,10) + RANDOM(1,10)
```

```
set stand to zero
```

```
while dealer_hand < 21 and your_hand < 21 and stand is zero:
```

```
...
```

```
endwhile
```

```
print dealer_hand
```

```
print your_hand
```

```
if dealer_hand == your_hand:
```

```
    print "It's a tie game!"
```

```
else if dealer_hand == 21 or your_hand >21:
```

```
    print "Dealer wins!"
```

```
else if your_hand == 21 or dealer_hand >21:
```

```
    print "You win!"
```

```
else if dealer_hand >your_hand:
```

```
    print "Dealer wins!"
```

```
else:
```

```
    print "You win!"
```

توضیح شبه‌کد:



زمانی که بازیکن دستور stand را وارد می‌کند به معنی اتمام دور است پس متغیر stand برای مشخص کردن وارد شدن دستور stand توسط بازیکن تعریف شده‌است تا تغییر آن را فراموش نشود. در ضمن Dealer یک کارت دیگر برای رونمایی در پایان بازی دارد اضافه کردن آن به دست را در پایان دور انجام دهید.

Botfather قصد دارد بازی را به نحوی پیاده سازی که پس از اتمام هر دور و مشخص شدن برنده، کاربر بتواند دور جدیدی را شروع نماید.



به نظر شما برای این منظور چه تغییراتی را در کد باید اعمال کند؟



برای ارتباط بیشتر با گُدخدا و Botfather می‌توانید سوالات خود را از طریق ایمیل از آن‌ها بپرسید.

Email...

Email...

;