Final Report

Sonic Pi Tutorials in Turkish

Hoşgeldiniz

Sonic Pi'ye hoş geldin. Umarım, sana göstereceğim çılgın sesleri çıkarmaya başladığın için heyecanlanırsınız. Müzik, sentez, programlama, kompozisyon, performans ve diğerleri hakkında çok şey öğreneceğin gerçekten eğlenceli bir yolculuk olacak.

Ama bekle, ne kadar kabayım! Kendimi tanıtmama izin ver - Ben Sam Aaron - Sonic Pi'yi yaratan adam. Beni Twitter'da @samaaron'da bulabilirsin ve sana merhaba demekten çok mutlu olurum. Ayrıca Sonic Pi ile kodladığım Canlı Kodlama Performanslarım hakkında daha fazla bilgi edinebilirsin.

Sonic Pi'yi geliştirmek için herhangi bir fikriniz varsa, lütfen bunları iletin, geri bildirim oldukça önemlidir. Asla bilemezsin, senin fikrin bir sonraki en önemli özellik olabilir!

Bu eğitim, kategorilere göre gruplandırılmış ve bölümlere ayrılmıştır. Baştan sona kadar kolay bir öğrenim için, uygun gördüğünüz bölümlere girip çıkmak size kalmış. Eksik bir şey olduğunu düşünüyorsanız bana bildirin, gelecekteki sürümler için değerlendireyim.

Son olarak, başkalarının canlı kodlamasını izlemek, öğrenmenin gerçekten harika bir yoludur. Düzenli olarak http://youtube.com/samaaron adresinden canlı yayın yapıyorum, bu yüzden lütfen tıklayın, merhaba deyin ve sorular sorun :-)

Tamam, hadi başlayalım…

Canlı Kodlama

Sonic Pi'nin en heyecan verici yönlerinden biri, tıpkı bir gitarla canlı olarak yapabileceğiniz gibi, müzik yapmak için canlı olarak kodlamanızı ve kodu değiştirmenizi sağlıyor. Bu bazı pratikler yapıldığında Sonic Pi'yi ile sahneye çıkabileceğiniz ve konsere katılabileceğiniz anlamına geliyor.

Zihninizi Boşaltın

Sonic Pi'nin bu eğitimin geri kalanında nasıl çalıştığının ayrıntılarına girmeden önce, size canlı kodun nasıl bir şey olduğunu deneyimlemeni istiyorum. Bundan çok (veya hiç) anlayamıyorsan endişelenme. Sadece tutunmaya çalış ve eğlen…

Canlı Döngü

Hadi sıradaki kodu boş tampona koyarak başlayalım:

```
live_loop :flibble do
 sample :bd_haus, rate: 1
 sleep 0.5
end
```

Şimdi Run butonuna bas, hızlı ve güzel bir davul vuruşu duyacaksın. Eğer sesi durdurmak istersen Stop butonuna basın. Yine de hemen başlatma, şu adımları takip et:

1- Davul vuruşunun devam ettiğinden emin ol
2- sleep değerini 0.5'ten daha yüksek (mesela 1) bir değerle değiştir.
3- Run butonuna tekrar bas
4- Davul vuruşunun hızının nasıl değiştiğini izle
5- Son olarak bu anı hatırla, bu senin Sonic Pi ile ilk canlı kodlama deneyimin, ve son olmayacak…

Tamam, bu yeterince kolaydı. Hadi karışıma başka birşey ekleyelim. `sample :bd_haus` 'un yukarısına `sample :ambi_choir, rate: 0.3` satırını ekle. Kodun şu şekilde gözükmeli:

```
live_loop :flibble do
 sample :ambi_choir, rate: 0.3
 sample :bd_haus, rate: 1
 sleep 1
end
```

Şimdi biraz kurcala. Oranları değiştir. Yüksek, alçak veya negatif değerler kullanınca ne oluyor? `:ambi_choir` için `rate:` değerini çok az (mesela 0.29) değiştirince ne oluyor? Çok fazla hızlı yaptığında bilgisayar hata vermeye başlıyor. Bu durumda beklemeyi yükselt ve tekrar dene.

Numune satırlarına # ile yorum atmayı dene:

```
live_loop :flibble do
 sample :ambi_choir, rate: 0.3
#  sample :bd_haus, rate: 1
 sleep 1
end
```

Bilgisayara onu görmezden gelmesini söyledin, dikkat et, bu yüzden o sesi duymuyoruz. Buna yorum denir. Sonic Pi'da, karışımdaki şeyleri kaldırmak ve eklemek için yorumları kullanabiliriz.

Son olarak, sana oynayacak eğlenceli bir şey bırakmama izin ver. Aşağıdaki kodu al ve yedek bir belleğe kopyala. Şimdi, iki döngü olduğunu anlamaya çalış - yani iki şey aynı anda devam ediyor. Şimdi, en iyi yaptığın şeyi yapın - deneyimle ve oyna. İşte bazı öneriler:

-Numune sesinde değişiklik görmek için mavi rate: değerlerini değiştirmeyi dene
-Bekleme sürelerini değiştir ve iki döngünün de farklı oranlarda çaldığını duy
-Numune satırındaki yorumu (#) kaldır ve tekrar gelen gitar sesinin tadını çıkar
-Mavi mix: lerden herhangi birini 0 ile 1 arasında değiştirmeyi dene

Run'a basmayı hatırla, döngünün diğer gelişinde değişimi duyacaksın. Kendini karışıklık içinde bulursan panik olma, sadece Stop'a bas, tampondaki kodu sil, temiz bir kopyasını yap ve tekrar dene. Hata yapmak öğrenmenin en kolay yoludur.

```
live_loop :guit do
```

```
 with_fx :echo, mix: 0.3, phase: 0.25 do
   sample :guit_em9, rate: 0.5
 end
#  sample :guit_em9, rate: -0.5
 sleep 8
end

live_loop :boom do
 with_fx :reverb, room: 1 do
   sample :bd_boom, amp: 10, rate: 1
 end
 sleep 8
end
```

Şimdi her konuda merakını giderene kadar oynamaya devam et, gitgide daha farklı şeyleri nasıl yapacağını merak etmeye başlayacaksın. Artık eğitimin devamını okumaya hazırsın.

Hadi ne bekliyorsun…

Oynayarak Öğrenmek

Sonic Pi sizi hem bilgisayar hem de müzik hakkında oyun ve deneyler yoluyla öğrenmeye teşvik ediyor. En önemli şey, senin eğlenmen ve bu esnada, nasıl kodlama yapıldığını istemeden öğreniyorsun.

Hata Diye Bir Şey Yoktur

Hazır buradayken, sana müzik kodlayarak geçmiş yıllarım boyunca öğrendiğim bir nasihat vereyim - hata diye bir şey yok, fırsatlar var. Bu, caz ile ilgili sıklıkla duyduğum bir şey ama canlı kodlama ile eşit derecede iyi işliyor. Ne kadar deneyimli olursan ol - tamamen başlangıç seviyesinden başlayarak deneyimli bir canlı kodlayıcıya kadar, tamamen beklenmedik sonucu olan bazı kodlar çalıştıracaksın. Delice gelebilir - bu durumda onları gerçekten çalıştırın. Tamamen şok edici ve bildiklerini değiştiricek gibi gelebilir. Bunun olması önemli değil - onunla ne yapacağın önemli. Sesi al, değiştir ve harika bir şeye dönüştür. Kalabalık vahşice artacak.

Basit Başla

Öğrenirken çılgın şeyler yapmak cazip geliyor. Fakat bu düşünceyi bekletmeli ve daha sonra ulaşacağın uzak bir hedef olarak görmelisin. Şimdilik, bunun yerine yazabileceğin basit şeyleri düşün, bu daha eğlenceli ve ödüllendirici olacaktır. Bu, kafandaki harika şeye atılan yalnızca küçük bir adım. Bu küçük adım hakkında bir fikrin olduğunda, onu dene, geliştir, onunla oyna ve sana verdiği yeni fikirleri de hayata geçir. Çok geçmeden eğlenmek ve gerçek anlamda ilerlemek seni epey meşgul edicek.

Ayrıca yaptıklarını başkalarıyla paylaştığından emin ol!

BRLEŞLER

Tamam, bu kadar giriş yeter - hadi biraz seslere bakalım.

Bu bölümde, brleşleri tetikleme ve manipüle etme yöntemlerini ele alacağız. Brleş, birleştirici için bir kısaltma, ses yaratan bir şeyleri tanımlamak için cafcaflı bir kelime. Tipik olarak brleşlerin kullanımı oldukça karmaşıktır - özellikle bir karmaşık kablolar gibi birbirine bağlanmış Eurorack modülleri gibi analog sentezler. Ancak, Sonic Pi size bu gücün çoğunu çok basit ve ulaşılması kolay bir şekilde verir.

Sonic Pi'nin arayüzünün sadeliğine hemen kanmayın. Eğer sana göreyse, çok karmaşık ses manipülasyonlarında derinleşebilirsin. Şapkanı tut da uçmasın…

İlk Biplerin

Şu koda bir göz at:

```
play 70
```

Her şeyin başladığı yer burası. Devam et, uygulamanın en üstündeki kod penceresine kopyalayıp yapıştır (Run butonunun altındaki büyük beyaz boşluk). Şimdi Run'a bas.

Bip!

Yoğun. Bir daha bas. Ve bir daha. Ve bir daha…

Vay, çılgınca, eminim bunu bütün gün yapabilirdin. Ama bekle, sen kendini sonsuz bipler içinde kaybetmeden, numarayı değiştirmeyi deneyelim:

```
play 75
```

Farkı duyabiliyor musun? Daha düşük bir sayı dene:

```
play 60
```

Yani düşük sayılar, düşük bipler çıkarıyor ve yüksek sayılar yüksek bipler. Aynı piyanodaki gibi, piyanonun sol tarafındaki notalar düşük notalardır (daha kalın) ve piyanonun yüksek tarafı (sağ taraf) daha yüksek (ince) sesler çıkarır. Aslında sayılar gerçekten piyanodaki notaları temsil ediyor. play 47 denince gerçekten piyanodaki soldan 47. Notayı ifade ediyor. Yani, play 48 denince bir sonraki (sağdaki) notaya basmış oluyorsun. Şöyle denk geliyor ki, 4. Oktavdaki C notasını 60 sayısı ile gösterebilirsin. Git ve oynat: play 60

Bu sana bir şey ifade etmiyorsa merak etme - ben ilk başladığımda bana da bir şey ifade etmedi. Şuan önemli olan küçük sayıların küçük bipler, büyük sayıların büyük bipler çıkardığını bilmen.

Akorlar

Bir notayı oynatmak oldukça eğlenceli, ama birkaçını aynı anda oynatmak daha da eğlenceli. Hadi dene:

```
play 72
play 75
play 79
```

Göz alıcı! Yani birden fazla play yazdığında, hepsini aynı anda oynatıyor. Kendin dene - hangi sayılar beraber güzel duyuluyor? Hangileri kötü duyuluyor? Deneyimle, keşfet ve kendin bul.

Melodi

Yani notaları ve akorları oynamak eğlenceli - ya melodiler? Ya bir notayı diğeriyle aynı anda değilde sonra oynatmak isteseydin? Bu çok kolay, sadece aralarına sleep koyman gerek.

```
play 72
sleep 1
```

```
play 75
sleep 1
play 79
```

Ne kadar aşk dolu, birazcık arpej. O zaman sleep 1 derken 1 ne anlama geliyor? 1 bekleme süresini temsil eder. Aslında 1 vuruş bekle demek, ama şimdilik bunu 1 saniye bekle olarak düşünebilirsin. Ya arpejimizi biraz daha hızlı yapmak isteseydik? Bu sefer sleep değerimizi azaltmamız gerekirdi. Mesela yarım, 0.5:

```
play 72
sleep 0.5
play 75
sleep 0.5
play 79
```

Nasıl daha hızlı oynattığını farket. Şimdi kendin dene, süreleri ve notaları değiştir.

Deneyebileceğin bir başka şey ise notaları play 52.3 veya play 52.63 gibi ara değerlerle oynatmak. Oynat ve eğlen.

Geleneksel Nota İsimleri

Biraz müzik notasyonu bilenler (bilmiyorsanız endişelenmeyin) melodileri sayılar yerine C veya F# gibi nota isimleri ile yazmak isteyebilirler. Sonic Pi onları da düşündü. Bunu deneyebilirsin.

```
play :C
sleep 0.5
play :D
sleep 0.5
play :E
```

Notaların önüne pembe : ‘yı koymayı unutma. Ayrıca nota isminden sonra sayıyı koyarak kaçıncı oktavda oynatacağını da belirleyebilirsin:

```
play :C3
sleep 0.5
play :D3
sleep 0.5
```

```
play :E4
```

Eğer bir notayı daha keskin yapmak istersen, sonuna bir s ekle, mesela play :Fs3, ve notayı daha düz yapmak istersen, sonuna b ekle, mesela play :Eb3.

Şimdi çıldır ve kendi müziğini yap.

Brleş Seçenekleri: Amp ve Pan

Hangi notayı çalacağınızı ya da hangi numuneyi tetikleyeceğinizi kontrol etmenize izin vermesinin yanı sıra, Sonic Pi, sesleri oluşturmak ve kontrol etmek için çeşitli seçenekler sunar. Bunların çoğu bu eğitimde ele alınacak, her biri için yardım sistemindeki kapsamlı belgeler var. Ancak, şuan için en yararlı iki tanesini sunuyorum: amp ve pan. İlk önce gerçekte hangi seçeneklerin olduğuna bakalım.

Seçenekler

Sonic Pi, brleşler için seçenekleri (ya da kısa süreliğine seçmeyi) destekler. Seçenek, duyduğunuz sesin yönlerini değiştiren veya kontrol eden çalmak isteğiniz kontrollerdir. Her brleşin, sesi hassas bir şekilde ayarlamak için kendi seçenekleri vardır. Ancak, amp: ve pan seçenekleri (başka bir bölümde ele alınmıştır) gibi birçok ses tarafından paylaşılan ortak seçenek grupları vardır.

Seçeneklerin iki ana bölümü vardır: isimleri (kontrolün adı) ve değerleri (kontrolü ayarlamak istediğiniz değer). Örneğin, peynir adında bir seçeneğe sahip olabilirsiniz ve bunu değerini 1 olarak ayarlamak isteyebilirsiniz.

Seçmeler, virgül kullanarak çalmak için arama listesine bakar ve ardından seçeneğin ismi gelir, mesela amp: Örneğin:

```
play 50, cheese: 1
```

Aralarına virgül koyarak birden fazla seçeneği kullanabilirsiniz:

```
play 50, cheese: 1, beans: 0.5
```

Seçeneklerin sırası önemli değil, mesela bu örnek aynı şey:

```
play 50, beans: 0.5, cheese: 1
```

Brleşler tarafından algılanmayan seçenekler (peynir ve fasulye gibi saçma olanlar!) program tarafından reddedilir.

Eğer yanlışlıkla aynı seçeneği iki kez farklı değerlerle kullanırsanız, sonuncu kazanır. Mesela fasulyeler burda 0.5 yerine 2 değerine sahip olacak:

```
play 50, beans: 0.5, cheese: 3, eggs: 0.1, beans: 2
```

Sonic Pi'deki çoğu şey seçenekleri kabul eder, öyüzden bunları öğrenip denemeye biraz vakit ayır. Hadi ilk seçeneğimiz amp: ile oynayalım:

Genişlik

Genişlik, bir sesin yüksekliğinin bilgisayarda bir temsilidir. Yüksek genişlik yüksek bir ses çıkarırken düşük genişlik kıyasla daha sessiz bir ses çıkarır. Sonic Pi, süre ve notları temsil etmek için sayılar kullandığı gibi, genişliği temsil etmek için de sayılar kullanır. 0 genişliği sessizdir (hiçbir şey duymazsın), 1 genişliği ise normal hacimdir. 2, 10, 100'e kadar daha yüksek bir genişlik bile artırabilirsin. Ancak, tüm seslerin genişliği çok yükseldiğinde, Sonic Pi'nin onları bastırmak için kompresör denen şeyi kullandığını unutmayın, çünkü bu kulaklarınız için çok yüksek olurdu. Bu genellikle sesi çatallı ve garip yapabilir. Bu nedenle, bastırmayı önlemek için düşük genişlikleri, yani 0 ve 0.5 aralığında kullanmayı dene.

Genişlet

Bir senin genişliğini değiştirmek için amp: kullanabilirsin. Mesela 0.5 kullanırsan yarı genişliğinde oynatmış olursun:

```
play 60, amp: 0.5
```

İki katı genişlikte oynatmak için 2 kullan:

```
play 60, amp: 2
```

Tabiki farklı genişlikler de kullanabilirsin:

```
play 50, amp: 0.1
sleep 0.25
```

```
play 55, amp: 0.2
sleep 0.25
play 57, amp: 0.4
sleep 0.25
play 62, amp: 1
```

Kaydırma

Bir başka eğlenceli seçenek ise pan: sesin stereo olarak kaydırılmasını sağlıyor. Sesi sola kaydırmak, sol hoparlörden duyacağın ve sağa kaydırmak, sağ hoparlörden duyacağın anlamına gelir. Değerlerimizde, tamamen solu temsil etmek için -1, merkezi temsil etmek için 0'ı, ve tamamen sağı temsil etmek için 1 kullanırız. Elbette, sesimizin tam konumunu kontrol etmek için -1 ile 1 arasındaki herhangi bir değeri kullanmakta özgürüz.

Hadi sol hoparlörden oynatalım:

```
play 60, pan: -1
```

Hadi şimdi de sağ hoparlörden oynatalım:

```
play 60, pan: 1
```

Son olarak merkezden oynatalım (varsayılan pozisyon)

```
play 60, pan: 0
```

Hadi şimdi git ve seslerin genişliğini değiştirip kaydırmanın tadını çıkar!

Brleşleri Değiştirmek

Şimdiye kadar oldukça eğlenceli bip sesleri çıkardık. Ancak, muhtemelen artık temel bip sesinden sıkılmaya başladın. Sonic Pi'nin sunduğu tek şey bu mu?Sadece bip sesi çalmaktan daha fazlası var mı? Evet var, ve bu bölümde Sonic Pi'nin sunduğu heyecan verici sesleri keşfedeceğiz.

Brleşler

Sonic Pi'nin birçok enstrümanı var, brleş birleştiricilerin kısaltması. Numuneler kaydedilmiş sesleri temsil ederken, brleşler onları nasıl kontrol ettiğinize göre yeni sesler çıkarabilir (Bunu eğitimin devamında daha detaylı açıklayacağız). Sonic Pi'nin brleşleri çok detaylı ve güçlüdür, onlarla oynarken çok ama çok eğlenebilirsiniz. Önce kullanacağımız brleşi seçmeyi öğrenelim.

Heyecan Verici saw 'lar ve prophet'ler

Saw dalgası oynatabileceğin eğlenceli seslerden biri:

```
use_synth :saw
play 38
sleep 0.25
play 50
sleep 0.25
play 62
sleep 0.25
```

Hadi bir başka deneyelim - prophet:

```
use_synth :prophet
play 38
sleep 0.25
play 50
sleep 0.25
play 62
sleep 0.25
```

Ya ikisini birleştirirsek? Önce biri sonra diğeri:

```
use_synth :saw
play 38
sleep 0.25
play 50
```

```
sleep 0.25
use_synth :prophet
play 57
sleep 0.25
```

Şimdi aynı anda:

```
use_synth :tb303
play 38
sleep 0.25
use_synth :dsaw
play 50
sleep 0.25
use_synth :prophet
play 57
sleep 0.25
```

Brleşleri Keşfetmek

Hangi brleşleri kullanabileceğini görmek için Sonic Pi'nin seçenekler listesine göz at. Orada 20'den fazla seçenek var. Bunlar benim favorilerim:

- :prophet
- :dsaw
- :fm
- :tb303
- :pulse

Şimdi biraz oyna ve brleşleri değiştirmenin müziğini nasıl etkilediğini gözlemle. Brleşleri aynı anda kullanmayı dene ve yeni sesler yarat.

NUMUNELER

Daha önceden kaydedilmiş sesleri kullanmak müzik oluşturmak için harika bir yöntemdir. Mükemmel hip-hop geleneğinde, biz bu önceden kaydedilmiş seslere

numuneler deriz. O zaman, mikrofonun hazırsa, git ve yağmurun tenteye çarpan nazik sesini kaydet, artık sen de bir numune yarattın.

Sonic Pi numuneler ile bir sürü keyifli şey yapmanı sağlar. Sadece 130 tane ortak kullanıma hazır numuneleri kullanarak değil, aynı zamanda kendininkini işlemene de izin verir. Hadi başlayalım…

Tetikleyici Numuneler

Bip sesleriyle oynamak sadece başlangıç. Çok daha eğlenceli olan bir şey ise önceden kaydedilmiş numuneleri tetiklemek. Hadi dene:

```
sample  :ambi_lunar_land
```

Sonic Pi çok fazla oynayabileceğin numune içeriyor. Bunları sadece play komutuyla kullanabilirsin. Birden fazla numune ve nota oynatmak için tek yapman gereken onları peş peşe yazmak:

```
play 36
play 46
sample :ambi_lunar_land
sample :ambi_drone
```

Eğer aralarında zaman boşluğu olmasını istiyorsan, sleep komutunu kullan:

```
sample :ambi_lunar_land
sleep 1
play 48
sleep 0.5
play 36
sample :ambi_drone
sleep 1
play 36
```

Sonic Pi'ın bir sonraki sesi oynatmak için bir sesin bitmesini beklemediğini farkettin mi? sleep komutu seslerin tetiklenmesinin ayrılmasını tanımlar. Bu senin sesleri kolayca katmanlara ayırmanı sağlamakla beraber, ilginç ses örtüşmeleri oluşturmanı da sağlar. Eğitimin devamında seslerin sürelerini nasıl kontrol edeceğimizi öğrenicez.

Numuneleri Keşfetme

Sonic Pi'ın sağladığı numuneleri keşfetmenin iki yolu var. Birincisi, yardım sistemini kullanabilirsin. Yardım penceresinin altındaki menüde Samples'a (numuneler) tıkla, kategorini seç ve uygun seslerin listesini göreceksin.

Alternatif olarak, otomatik tamamlama sistemini kullanabilirsin. Basitçe numunenin başını şu sekilde yazabilirsin: sample :ambi_ , ardından aralarından seçmen için diğer numune isimlerinin gözüktüğü bir pencere açılacak. Şu kategori ön eklerini deneyebilirsin:

:ambi_
:bass_
:elec_
:perc_
:guit_
:drum_
:misc_
:bd_

Şimdi eserindeki numuneleri birleştirmeye başlayabilirsin!

Numune Parametreleri: Amp ve Pan

Brleşlerde gördüğümüz gibi sesleri parametreler ile kontrol edebiliyoruz. Numuneler parametrelendirme mekanizmasını tamamen destekliyor. Hadi amp: ve pan: 'a bir göz atalım.

Numuneler ile amp

Numunelerin genişliğini brleşlerdeki ile tamamen aynı şekilde değiştirebilirsiniz.

```
sample :ambi_lunar_land, amp: 0.5
```

Numuneler ile pan

Numunelerde aynı zamanda `pan:` parametresini de kullanabiliriz. Mesela, bu örnek sesin yarısını sol taraftan, sonrasında da kalan yarısını sağ taraftan nasıl oynatabileceğimizi gösteriyor.

```
sample :loop_amen, pan: -1
sleep 0.877
sample :loop_amen, pan: 1
```

Şunu unutmayın ki 0.877 `:loop_amen` 'in saniye cinsinden süresinin yarısıdır.

Son olarak şunu da eklemeliyim ki brleşleri `use_synth_defaults` ile oynatırsanız (eğitim devamın da daha detaylı tartışacağız), brleşler numune tarafından göz ardı edilecektir.

Numuneleri Esnetmek

Şu ana kadar çeşitli brleşler ve numuneler oynatmayı öğrendik, şimdi bunları daha eşsiz ve ilginç bir şekilde nasıl modifiye edebileceğimizi öğrenmenin vakti geldi. Önce numuneleri esnetme ve sıkıştırma nasıl yapılır onu göreceğiz.

Numune gösterimi

Numuneler, sesi yeniden üretmek için hoparlör konisinin nasıl hareket oynatıldığını temsil eden numaralar olarak kaydedilmiş önceden kaydedilmiş seslerdir. Hoparlör konisi içeri ve dışarı hareket edebilir, bu yüzden sayılar koninin her an için ne kadar içeri girmesi gerektiğini göstermelidir. Kayıtlı bir sesi güvenilir bir şekilde çoğaltmak için, numunenin saniyede binlerce numara kaydetmesi gerekir! Sonic Pi, bu sayıların listesini alır ve bilgisayarınızın hoparlörünü sesi çıkarmak için doğru şekilde hareket ettirmek için kullanır. Ancak, sesi değiştirmek için sayıların hoparlöre verilme hızını değiştirmek de eğlencelidir.

Oran Değişimi

Hadi ortamdaki seslerden birini oynatalım: `:ambi_choir` .Normal değerle oynatmak için `sample:` 'da ki `rate:` 'i değiştirebilirsiniz.

```
sample :ambi_choir, rate: 1
```

Bu normal hızda oynatır (1), bu yüzden henüz özel bir şey yok. Ancak, bu numarayı başka bir şeyle değiştirmekte özgürüz. 0.5 nasıl:

```
sample :ambi_choir, rate: 0.5
```

Vay! Burada neler oluyor? Peki, iki şey. İlk olarak, örnek oynamak için iki kat daha uzun sürüyor, ikinci ses ise bir oktav düşüktür. Bunları biraz daha detaylı inceleyelim.

Esnetme Vakti

Uzatmak ve sıkıştırmak için eğlenceli bir örnek Amen Break. Normal oranda, bir drum'n'bass bas parçasına atmayı düşünebiliriz:

```
sample :loop_amen
```

Ancak oranı değiştirerek türleri değiştirebiliriz. old school hip-hop için yarı hızını deneyin:

```
sample :loop_amen, rate: 0.5
```

Bunu hızlandırırsak, jungle alanına girmiş oluruz.

```
sample :loop_amen, rate: 1.5
```

Son parti hilemiz ise negatif oran kullanmak:

```
sample :loop_amen, rate: -1
```

Vay! Geri oynatıyor! Şimdi farklı oranlarda birçok farklı numuneyle oynamayı deneyin. Çok hızlı oranları deneyin. Çılgın yavaş oranları deneyin. Hangi ilginç sesleri elde edebileceğinizi görün.

Numune Oranının Basit Bir Açıklaması

Numuneleri düşünmenin yararlı bir yolu yaylardır. Oynatma hızı, yayı sıkmak ve germek gibidir. Numuneyi 2 hızında oynatırsanız, yayı normal uzunluğunun yarısı kadar sıkıştırıyorsunuz. Bu nedenle numune daha kısa olduğu için oynama süresinin yarısını alır. Numuneyi yarı hızda oynatırsanız, yayı uzunluğunu ikiye katlayacak şekilde uzatırsınız. Bu nedenle örnek, daha uzun süre oynamak için iki kat zaman alır. Ne kadar sıkarsanız (daha yüksek oranda), o kadar kısalır, o kadar fazla gerilir (daha düşük oranda), o kadar uzun sürer.
Bir yayın sıkıştırılması yoğunluğunu arttırır (cm başına bobin sayısı) - bu durum daha yüksek perdeli ses çıkmasına benzer. Yayı germek yoğunluğunu azaltır ve daha düşük perdeli sese benzer.

Sarmaşık Numuneler

Bir ADSR zarfı kullanarak bir numunenin süresini ve genişliğini değiştirmek de mümkündür. Bununla birlikte, bu, brleşlerde mevcut olan ADSR zarfından biraz farklı şekilde çalışır. Numune zarfları, yalnızca numunenin genişliğini ve süresini azaltmanıza izin verir - ve asla arttırmayın. Numune çalmayı bitirdiğinde veya zarf tamamlandığında, hangisi önce olursa, numune duracaktır. Bu nedenle, çok uzun bir `release:`

kullanıyorsanız :, numunenin süresini uzatmaz.

Amen Zarfları

Hadi güvenilir arkadaşımız Amen Break'e bakalım.

```
sample :loop_amen
```

Seçeneği olmadan, tam numuneyi tam genlikte duyarız. Bunu 1 saniyeden fazla bir sürede solmak istiyorsak `attack:` 'ı kullanabiliriz.

```
sample :loop_amen, attack: 1
```

Daha kısa bir karartma için daha kısa bir saldırı değeri seçin.

```
sample :loop_amen, attack: 0.3
```

Otomatik Sürdürme

ADSR zarfının davranışının standart brleş zarfından farklı olduğu durumlarda sürdürme değerindedir. Standart brleş zarfında, siz manuel olarak ayarlamadıysanız sürdürme varsayılan olarak 0 olarak ayarlanmıştır. Numunelerde, sürdürme değeri varsayılan olarak otomatik bir değere döner - numunenin geri kalanını oynatmak için kalan süre. Bu yüzden hiçbir yanılgıya düşmediğimizde tam numuneyi duyarız. Eğer saldırı, çürüme, devam etme ve bırakma değerleri 0 olsaydı, asla gözetleme sesi duymazdık. Sonic Pi bu nedenle numunenin ne kadar sürdüğünü hesaplar, herhangi bir saldırıyı keser, çürüme ve serbest bırakma sürelerini hesaplar ve sonucu sürdürme süresi olarak kullanır. Saldırı, çürüme ve serbest bırakma değerleri numunenin süresinden daha fazla eklerse, süreklilik sadece 0 olarak ayarlanır.

Soldurma

Bunu anlamak için Amen Break'imizi daha detaylı inceleyelim. Sonic Pi'ye numunemiz ne kadar uzun olsun diye soracak olursak eğer:

```
print sample_duration :loop_amen
```

Numunenin saniye cinsinden uzunluğu olan `1.753310657596372` yazdıracaktır. Buradaki rahatlığı için onu `1.75`'e çıkaralım. Şimdi, serbest bırakmayı `0.75` olarak ayarlarsak, şaşırtıcı bir şey olur:

```
sample :loop_amen, release: 0.75
```

Numunenin ilk saniyesini tam genlikte oynatır, daha sonra 0.75 saniyelik bir sürede söner. Bu, eylemdeki otomatik sürdürmedir. Varsayılan olarak, sürüm her zaman numunenin sonundan çalışır. Numunemiz 10.75 saniye uzunluğunda olsaydı, 0.75 saniyeden fazla solmadan önce ilk 10 saniyeyi tam genişlikte oynatırdı.
Unutmayın: varsayılan olarak, `release:` numunenin sonunda kaybolur.

Belirgin Sürdürme

`sustain:`'i el ile ayarlayarak normal brleş ADSR davranışımıza kolayca geri dönebiliriz: 0 gibi bir değer:

Şimdi, numunelerimiz toplamda sadece 0,75 saniye çalıyor. `attack:` için varsayılan: ve `decay:` 0'da, numune doğrudan tam genişliğe atlar, orada 0 saniye bekler, sonra bırakma periyodu boyunca tekrar 0 genişliğe kadar serbest kalır - 0,75 sn.

Vurma Zilleri

Bu davranışı, daha uzun sesli numuneleri daha kısa, daha vurgulu versiyonlara dönüştürmek için iyi bir etki olarak kullanabiliriz. Numuneye göz atın:

```
:drum_cymbal_open:
```

```
sample :drum_cymbal_open
```

Şimdi gidip numunelerin üzerine zarf koyarak eğlenin. Gerçekten ilginç sonuçlar için oranı da değiştirmeyi deneyin.

Kısmi Numuneler

Bu bölüm, Sonic Pi'nin numune oyuncusunu incelememizin sonudur. Hızlıca özetleyelim. Şimdiye kadar numuneleri nasıl tetikleyebileceğimize baktık:

```
sample :loop_amen
```

Daha sonra numunelerin oranını nasıl yarı hızda çalma gibi değiştirebileceğimize baktık:

```
sample :loop_amen, rate: 0.5
```

Daha sonra, bir numuneyi nasıl solda alabileceğimize baktık (hadi yarı hızda yapalım):

```
sample :loop_amen, rate: 0.5, attack: 1
```

Ayrıca, bir numunenin başlangıcını, sürekli bir değer vererek ve hem saldırının hem de serbest bırakmanın kısa değerler olarak belirleyerek vuruşları nasıl kullanabileceğimize baktık:

```
sample :loop_amen, rate: 2, attack: 0.01, sustain: 0,
release: 0.35
```

Ancak, her zaman numunenin başında başlamamız gerekmeseydi hoş olmaz mıydı? Numunenin her zaman sonunda bitirmek zorunda kalmasak iyi olmaz mıydı?

Başlangıç Noktası Belirleme

Numunede isteğe bağlı bir başlangıç noktası seçerek 0 ile 1 arasında bir değer vermek mümkündür, burada 0, numunenin başlangıcı, 1, son ve 0,5 numunenin yarısı kadardır. Amen break'in sadece son yarısını oynamayı deneyelim:

```
sample :loop_amen, start: 0.5
```

Ya numunenin son çeyreğini denersek?

```
sample :loop_amen, start: 0.75
```

Bitiş Noktası Belirleme

Benzer şekilde, numunedeki bir bitiş noktasını 0 ile 1 arasında belirlemek mümkündür.

```
sample :loop_amen, finish: 0.5
```

Başlangıç ve Bitişi belirleme

Elbette, bu ikisini ses dosyasının isteğe bağlı bölümlerini oynatmak için birleştirebiliriz. Ortadaki sadece küçük bir bölüm için:

```
sample :loop_amen, start: 0.4, finish: 0.6
```

Ya başlangıç noktasını bitiş noktasından sonra seçersek ne olur?

```
sample :loop_amen, start: 0.6, finish: 0.4
```

Havalı! Geriye doğru oynatıyor!

Combining with rate

Son olarak, bütün bunları ADSR zarfımızla ilginç sonuçlar verecek şekilde birleştirebiliriz.

```
sample :loop_amen, start: 0.5, finish: 0.8, rate:
-0.2, attack: 0.3, release: 1
```

Harici Numuneler

Yerleşik numuneler hızlı bir şekilde başlamanızı sağlarken, müzikteki diğer kaydedilmiş sesleri denemek isteyebilirsiniz. Sonic Pi bunu tamamen destekliyor. İlk önce, eserinizin taşınabilirliği hakkında hızlıca tartışalım.

Taşınabilirlik

Parçanızı tamamen yerleşik brleş ve numunelerle oluşturduğunuzda, müziğinizi yeniden üretmek için gereken tek şey kodlamaktır. Bir an için düşününce bu oldukça şaşırtıcı! Gist'in çevresine e-postayla gönderebileceğiniz veya bağlı kalabileceğiniz basit bir metin parçası, seslerinizi yeniden oluşturmak için ihtiyacınız olan her şeyi sağlar. Bu kodu arkadaşlarınızla paylaşmayı gerçekten kolaylaştırır çünkü sadece kodu yapıştırmanız gerekir.

Ancak önceden kaydedilmiş kendi numunelerinizi kullanmaya başlarsanız, bu taşınabilirliği kaybedersiniz. Bunun nedeni, müziğinizi çoğaltmak için başkalarının yalnızca kodunuza değil, örneklerinize de ihtiyaç duymasıdır. Bu, başkalarının

çalışmanızı manipüle etme, ezme ve deney yapmasını engeller. Elbette bu kendi numuneleri kullanmanızı engellememelidir, sadece bu konuda dikkatli olmalısınız.

Yerleşik Numuneler

Rastgele bir WAV, AIFF veya FLAC'ı bilgisayarınızda nasıl oynatabilirsiniz? Tek yapmanız gereken o dosyanın yolunu numuneye geçirmek:

```
# Raspberry Pi, Mac, Linux
sample "/Users/sam/Desktop/my-sound.wav"
# Windows
sample "C:/Users/sam/Desktop/my-sound.wav"
```

Sonic Pi otomatik olarak numuneyi yükler ve oynatır. Ayrıca, tüm standart parametreleri de geçirebilirsiniz:

```
# Raspberry Pi, Mac, Linux
sample "/Users/sam/Desktop/my-sound.wav", rate: 0.5,
amp: 0.3
# Windows
sample "C:/Users/sam/Desktop/my-sound.wav", rate: 0.5,
amp: 0.3
```

Randomizasyon

Rastgele numaralar eklemek müziğinizi çekici kılmak için çok güzel bir yöntemdir. Sonic Pi'nin somut işlevselliği bu randomizasyonu müziğinize eklemenizi sağlar. Fakat önce şok edici gerçeği öğrenelim. Sonic Pi'nin randomizasyonu aslında random değildir. Bu nedemek? Hadi görelim.

Tekrarlanabilirlik

rrand çok kullanışlı bir fonksiyondur. Size iki sayı arasından –min ve max rastgele bir sayı dönmesini sağlar. Hadi random bir nota çalmayı deneyelim.

```
play rrand(50, 95)
```

Ooh, rastgele bir nota çaldı. 83.7527 çalındı. 50 ve 95 arasında ranstgele bir nota. Bekle az önce seninle aynı notayı mı tahmin ettim? Burda garip bir şeyler oluyor. Hadi tekrar dene. Yine mi 83.7527 yi seçti? Bu gerçek bi randomizasyon olamaz!
Cevapta tam olarak bu. Aslında bu gerçek bir randomizasyon değil. Sonic Pi sistemi sana rastgele bir notayı tekrarlanan bir şekilde verir. Bu sana sende çalan müziğin başka bir sistemde de aynı şekilde duyulma olanığını sağlar.

Tabiki verilen örnekte sistem herseferinde “rastgele” 83.7527yi seçti. Bu ilginç olurda fakat aslında değil. Şunu bir dene:

```
loop do
  play rrand(50, 95)
  sleep 0.5
end
```

Sonunda bu gerçekten random duruyor. Bu örnekte sürekli çağırılan randomizasyon fonksiyonu her seferinde rastgele değerler döndürecek. Fakat sonraki çalıştırmada aynı değerleri döndürecek.

Perili Çanlar

Randomizasyonun çok güzel bir örneği sürekli :perc_bell örneğini rasgele hızlarda çalan perili çanlardır.

```
loop do
  sample :perc_bell, rate: (rrand 0.125, 1.5)
  sleep rrand(0.2, 2)
end
```

Rastgele kesme

Randomizasyona bir başka eğlenceli örnek:

```
use_synth :tb303

loop do
  play 50, release: 0.1, cutoff: rrand(60, 120)
```

```
  sleep 0.125
end
```

choose

choose fonksiyonu sürekli kullanılan ve verilen element listesinden rasgele bir eleman seçmeye yarayan fonksiyondur. Örnek vermek gerekirse ben 60 65 veya 72 notalarından birinin oynatılmasını isteyebilirim. Bunu sağlamak için choose fonksiyonunu kullanmam lazım. Öcenlikle oynatılmasını isteyebileceğim notaları köşeli parantez içine yazmam ve virgüller ile ayırmam lazım [60,65,72]. Sonrasında choose fonksiyonunu çağırmak gerek.

```
choose([60, 65, 72])
```

Hadi neye benzediğini duyalım:

```
loop do
  play choose([60, 65, 72])
  sleep 1
end
```

rrand

Aslında rrand fonksiyonunu daha önce gördük ama üstünden bir kez daha geçelim. Rrand bana iki sayı arasından rastgele bir değer döndürür. Bu demek oluyorki seçilen iki sayı arasından o iki sayıyı asla döndürmez bunun yerine aradaki sayıları tercih eder. Rrand(20,110) dan döndürülebilecek sayılar:

87.505

86.0525

61.778

rrand_i

yukardaki gibi virgüllü bir sayı değilde bir tam sayı elde etmek istenebilir. rrand_i(20,110) içinden döndürülebilecek sayılar:

88

86

82

rand

Bu fonksiyon size minimum değer olan 0 ile maksimum değer olan 1 arasından bir değer döndürülmesini sağlar bu amp: kullanırken çok işlevseldir.

```
loop do
  play 60, amp: rand
  sleep 0.25
end
```

rand_i

Bu ise rrand ve rrand_i fonksiyonları arasındaki ilişkiye benzer olarak 0 ile 1 arasından rastgele bir sayı döndürülmesini sağlar.

Programming Structures

Şimdiye kadar ses oluşturmak için kullanılan play ve sample konularının ve melodiyi yaratmak için yazdığınız sleep komutunun temellerini öğrendiniz. Şimdi bu programın size başka neler sunabileceğini merak ediyorsunuz… döngüler, şartlılar,iplikler gibi temel orgramlama yapıları müziğinize yaratmak için çok güçlü araçlardır. Hadi temeller ile başlayalım

5.1 Bloklar

Bu yapıyı Sonic Pi de oldukça fazla göreceksiniz. Bloklar bize çok sayıda kod kullanılan müziklerde bir çok kullanışlı işi yapmamızı sağlar. Örneğin sytnh ve sample parametreleri bize tek bir satırda bazı şeyleri değiştirme olanağı verir. Fakat bazen biz satırlarca kod kullanarak anlamlı şeyler yaratmak isteriz(dögüye almak gibi). Aşşağıdaki kodu ele alalım:

```
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
```

Bu kadar kodu doğru bir şekilde yazabilmek için Sonic Pi sistemine bizim kodun nerde başladığını ve nerde bittiğini söylememiz lazım. Bunun için do ve end kullanımı şart:

```
do
  play 50
```

```
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

5.2 Tekrarlama ve Döngüler

Şimdiye kadar zamanımızın çoğunu play sample ve sleep kullanımı öğrenerek geçirdik. Sizinde farkettiğiniz üzere bloklar kullanarak bir çok eğlecenli şey yapabiliriz. Tekrarlama ve döngüler ile başlayalım.

Repetition

Birkaç kez tekrarlayan bir kodu nasıl yazarsınız. Diyelimki böyle bir kodunuz var

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

Bunu 3 kere oynatmak istiyoruz:

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25

play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25

play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
```

```
sleep 0.25
```

Buradaki en önemli sorun verilen kodu 3 kere değilde 50 kere ve hatta 1000 kere oynatmak istediğiniz zaman bunu herseferinde kopyala yapıştır olarak yapamamanız. Üstelik bi yeri değiştirmek istediğiniz zaman bunu her bir kod için tekrarlanamanız lazım.

tekrarlama

Sonuç olarak belli bir kodu tekrarlamak istediğiniz zaman bunu kolayca 3 kere yap demelisiniz. Eski dostumuz blokları hatırlıyormusunuz? Start ve end bloklarını istenilen kodu 3 kere oynatmak için kısaca 3. Times do yazarak yapabilirsiniz:

```
3.times do
  play 50
  sleep 0.5
  sample :elec_blup
  sleep 0.5
  play 62
  sleep 0.25
end

4.times do
  play 50
  sleep 0.5
end

8.times do
  play 55, release: 0.2
  sleep 0.25
end
```

Döngülemek

Bazen kendizi 1000 gibi büyük sayıları tekralamk isterken bulabilirsinz. Böyle bir durumda Sonic Pi ye bu melodiyi sürekli çal diyebilmeniz sizin hakkınız. Hadi bunu uygulayalım

```
loop do
  sample :loop_amen
  sleep sample_duration :loop_amen
end
```

Bunun ile ilgili en önemli şey eğer kodunuz sonsuz bir döngünün içine girerse bu karadelik görevi görebilir. Bir başka deyişle kodunuz o döngünün içinde sıkışabilir. Bu örnekte cymbal asla çalmayacak.

```
loop do
  play 50
  sleep 1
end

sample :drum_cymbal_open
```

5.3 Şartlılar

eğer sadece rastgele bir nota değilde kodunuzu dayanak alan rastgele bir karar verilmesini istiyorsanız if kullanımı yapabilirsiniz.

Yazı Tura

Hadi yazı tura oynayalım. Yazı gelirse drum. Tura gelirse cymbal çalsın. Bunu one_in fonksiyonu ile sağlayabiliriz. Bu fonksiyondan çıkacak sonucu ise drum ve ya cymbal dan birisini seçmek için kullanabiliriz:

```
loop do

  if one_in(2)
        sample :drum_heavy_kick
  else
        sample :drum_cymbal_closed
  end

  sleep 0.5

end
```

Unutma if koşulunun 3 bölümü var:

- Sorulacak soru
- İlk tercihte uygulanacak kod
- İkinci tercihte uygulanacak kod

Bunu saplamak için if else yapıları kullanılabilir

```
loop do
```

```
  if one_in(2)
        sample :drum_heavy_kick
        sleep 0.5
  else
        sample :drum_cymbal_closed
        sleep 0.25
  end

end
```

Basit if

Bazen duruma bağlı oalrak tek satırlık kod çalıştırmak isteyebilirsiniz. Bu if yapısını konumlandırarak mümkün olarabilir:

```
use_synth :dsaw

loop do
  play 50, amp: 0.3, release: 2
  play 53, amp: 0.3, release: 2 if one_in(2)
  play 57, amp: 0.3, release: 2 if one_in(3)
  play 60, amp: 0.3, release: 2 if one_in(4)
  sleep 1.5
end
```

5.4 İplikler

Bass kodunu ve phat beatini yarattın. Peki bunları nasıl aynı anda oynatabilirsin? İlk çözüm olarak herbirini manuel olarak kodlayabilirsin- biraz bass biraz davul sonra daha fazla bass… Fakat hepsini mauel olarak yapmak çok zahmetli ve zor bir iş. Sonic Pi sana yeni bir çözüm sunuyor iplikler.

Sonsuz döngü

Bir örneği inceleyelim:

```
loop do
  sample :drum_heavy_kick
  sleep 1
end

loop do
  use_synth :fm
```

```
  play 40, release: 0.2
  sleep 0.5
end
```

Önceden konuştuğumuz gibi bu döngü sonsuz bir döngü içinde girebilir buda programının sonraki koda geçmesini engeller ve bir kara deliğin içine çeker. Böyle bir durumda iplikler seni kurtarmak için gelir.

Kurtarmak için iplikler

```
in_thread do
  loop do
        sample :drum_heavy_kick
        sleep 1
  end
end

loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end
```

in_thread do bloğu sayesinde sonic Pi sistemine kara delikten çıkması için yardım edebiliyoruz.

Peki ya üste yeni bir synth eklemek istersek:

```
in_thread do
  loop do
        sample :drum_heavy_kick
        sleep 1
  end
end

loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end

loop do
  use_synth :zawa
  play 52, release: 2.5, phase: 2, amp: 0.5
```

```
  sleep 2
end
```

İplik şeklinde oynatma

Sizi şaşırtabilecek bir başka şey ise oynatma tuşuna basıldığında aslında size yeni bir iplik yaratıp oynatıyorsunuz. Bu yüzden bir iplik oynaması sırasında tekrar oynatma tuşunu basılırsa üst üste binmiş sesler ortaya çıkıyor.

Miras

İn_thread ile yeni bir iplik oluşturduğunda yeni iplik mevcut olan bütün ayarları üstlenir. Hadi görelim:

```
use_synth :tb303
play 50
sleep 1

in_thread do
  play 55
end
```

İplik isimlendirme

Sonunda ipliğimizi isimlendirebiliriz

```
in_thread(name: :bass) do
  loop do
        use_synth :prophet
        play chord(:e2, :m7).choose, release: 0.6
        sleep 0.5
  end
end

in_thread(name: :drums) do
  loop do
        sample :elec_snare
        sleep 1
  end
end
```

Her bir ipliğin kendine has bir ismi olmalıdır

Bilinmesi gerekn son şey ise bir seferde aynı isimdeki ipliklerden sadece birisi oynatılır:

```
in_thread do
  loop do
        sample :loop_amen
        sleep sample_duration :loop_amen
  end
```

```
end
```
Devam et ve müziğine tampon ekle ve oynatma tuşuna bas daha sonra oynatma tuşunu ard arda birkaç kez daha bas. Şimdi durdur. Bu davranışı daha önce sürekli gördün. Oynatma tuşuna basınca sesler üst üste biniyor fakat farklı isimli iplikler olunca:

```
in_thread(name: :amen) do
  loop do
        sample :loop_amen
        sleep sample_duration :loop_amen
  end
end
```

5.5 Fonksiyonlar

Kodlamaya başladığınızda yazılan kodları organize etmen gerekiyor.

Fonksiyon tanımlamak

```
define :foo do
  play 50
  sleep 1
  play 55
  sleep 2
end
```

Burada foo isimli yeni bir fonksiyon oluşturduk. Fark ettiysen bunu do ve end bloklarının içinde yaptık ve sihirli kelime define ı kullandık.

Fonksiyon çağırmak

Fonksiyonları sadece isimlerini yazarak çağırabiliriz:

```
define :foo do
  play 50
  sleep 1
  play 55
  sleep 0.5
end

foo
sleep 1
2.times do
  foo
end
```

Parametereleştirilmiş fonksiyonlar

Parametrelerin do ve define bloklarının arkasından gelmesi gerekiyor, aynı zamanda | ile çevrilme ve , ile ayrılmalı. Parametreleri birer söz oalrak görebilirsiniz hepsi en son değerleri ile yer değiştirecek. İlginç bir öreneği inceleyelim:

```
define :chord_player do |root, repeats|
  repeats.times do
        play chord(root, :minor), release: 0.3
        sleep 0.5
  end
end

chord_player :e3, 2
sleep 0.5
chord_player :a3, 3
chord_player :g3, 4
sleep 0.5
chord_player :e3, 3
```

Burada repeats i repeats.times do fonksiyonunun sayısı olarak kullandım. Aynı zamanda root uda nota ismi olarak play i çağırmak için kullandım.

5.6 Değişkenler

çok kullanışlı bir diğer şey ise kodunun elementleri için isimler ürtmek. Sonic Pi bunu çok kolaykaştırıyor, istediğiniz ismi yazıp = kullanarak yapabiliyorsunuz:

```
sample_name = :loop_amen
```

Anlamlarıyla iletişim

Bir kod yazarken kısaca bilgisayar ne yapmasını istediğini söylediğini düşünebilirsin. Bunu bilgisayar anladığı sürece her şey yolunda ve ilerde birisi veya sen kendi yazdığın kodu okuyabilrsin bu sebeple gelecekteki sen içinde kodunun okunabilir olması önemli. Bunu yapmanın bir yolu yorum eklemek diğeri ise anlamlı değişken isimleri kullanmak

```
sleep 1.7533
```

neden bu sayı 1.7533? Bu sayı nereden geldi? Ne anlama geliyor? Fakat birde şunu incele:

```
loop_amen_duration = 1.7533
sleep loop_amen_duration
```

Tekrarlamayı kontrol etmek

Kodunuzu değiştirmek istediğiniz zaman sıklıkla kodunuzda çok fazla sayıda tekrarlama görürsünüz bu sebeple çok fazla şeyin yerini değiştirmeniz gerekir.

```
sample :loop_amen
```

```
sleep sample_duration(:loop_amen)
sample :loop_amen, rate: 0.5
sleep sample_duration(:loop_amen, rate: 0.5)
sample :loop_amen
sleep sample_duration(:loop_amen)
```

:loop_amen ile çok fazla şey yaparken!! Eğer ki :loop_garzul'un sesini duymak istersen ne olur? Bütün :loop_amens ve loop_garzul ların yerini değiştirmemiz gerekir eğer çok fazla zamanınız varsa bu çok sorun değil. Fakat bazen zaman konusunda sıkışık olabiliriz. Ya kodumuz bu şekilde olursa:

```
sample_name = :loop_amen
sample sample_name
sleep sample_duration(sample_name)
sample sample_name, rate: 0.5
sleep sample_duration(sample_name, rate: 0.5)
sample sample_name
sleep sample_duration(sample_name)
```

Uyarı: değişkenler ve iplikler

```
a = (ring 6, 5, 4, 3, 2, 1)

live_loop :shuffled do
  a = a.shuffle
  sleep 0.5
end

live_loop :sorted do
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
end
```

Yukarıdaki örnekte a değerine ring sayıları verdik.ve bunları 2 farklı canlı döngüde kullandık ilkinde 0.5 ve düzenli(ring 1,2,3,4,5,6) ve bunları bastırdık. Eğer oynatma tuşuna basarsan bu listenin aslında dizili olmadığını göreceksin. Bununiçin kullanabilecğin bazı çözümler var. İlk olarak aynı değişekeni farklı live_loop lar içinde kullanma. Şimdiki örnekte ne demek istediğimi daha iyi anlayacaksın:

```
live_loop :shuffled do
  a = (ring 6, 5, 4, 3, 2, 1)
  a = a.shuffle
  sleep 0.5
```

```
end

live_loop :sorted do
  a = (ring 6, 5, 4, 3, 2, 1)
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
end
```

5.7 İplik senkrpnizasyonu

canlı kodlamada ve çok sayıda konfsiyonun kullandımında ileri düzey pratikliğe ve seviyeye ulaştığın zaman fark edeceksinki bu kodlamalar esnasında hata yapmak aslında son derece kolay. Fakat bu büyük bi sorun değil tabiki çünkü ipliği yeniden başlatabilirsin. Fakat ipliği yeniden başlattığında mevcut iplikle karışma ihtimali var.

İşaret ve Senkronizasyon

Sonic Pi yine sana bir çözüm yolu sunuyor, Cue ve Sync
Cue diğer ipliklere kalpatışı mesajı göndermeni sağlar. Mevcut ayarlara göre diğer iplikler bu mesajları önemsemezler, sync burdaa devreye giriyor.
Hadi detayları keşfedelim:

```
in_thread do
  loop do
        cue :tick
        sleep 1
  end
end

in_thread do
  loop do
        sync :tick
        sample :drum_heavy_kick
  end
end
```

Sonuç olarak :drum_heacy_kick sesini duyuyoruz diğer iplik :tick mesajını yolladığı zaman.(iplikler aynı anda başlamasalar bile)

```
in_thread do
  loop do
        cue :tick
        sleep 1
```

```
  end
end

sleep(0.3)

in_thread do
  loop do
        sync :tick
        sample :drum_heavy_kick
  end
end
```

Cue isimleri

Hadi biraz cue isimleri ile oynayalım:

```
in_thread do
  loop do
    cue [:foo, :bar, :baz].choose
        sleep 0.5
  end
end

in_thread do
  loop do
    sync :foo
    sample :elec_beep
  end
end

in_thread do
  loop do
        sync :bar
        sample :elec_flip
  end
end

in_thread do
  loop do
        sync :baz
        sample :elec_blup
```

```
  end
end
```

6.1 FX ekleme

Bu bölümde bazı çift FX: reverb ve echo ları inceleyeceğiz. Sonic Pi nin FX sistemi blokları dahil etmeyi ilgilendiriyor 5.1 i incelemediyseniz tam zamanı.

Yankı

Eğer yankı kullanmak istiyorsanız wirth_fx :reverb yazın

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Eğer do end bloğunun dışında da kodumuz varsa:

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end

sleep 1
play 55
```

Fark ettiyseniz play 55 yankılı bir şekilde oynatılmadı çünkü ilgili kod do end bloğunun dışarısında.

```
play 55
sleep 1

with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

```
sleep 1
play 55
```

Eko

Seçilebilecek bir sürü FX var peki ya eko?

```
with_fx :echo do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Hadi ekoyu hızlandıralım

```
with_fx :echo, phase: 0.125 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Aynı zamanda deca: kullanarak ekonun süresini uzatalım(zaman 8 kere çalınacak şekilde ayarlansın):

```
with_fx :echo, phase: 0.5, decay: 8 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

6.2 Uygulamada FX

FX kullanımını hernekadar anlattıysakta aslında FX ler kullanım koşunda son derece komplex yapılardır. Kolay görünümleri sıklıkla insanların parçalarını fazla kullanmasına yol açıyor. Bu kodu inceleyelim:

```
loop do
  with_fx :reverb do
        play 60, release: 0.1
        sleep 0.125
  end
end
```

bu kodda 60. Nota yı oynatıyoruz ve kısa bir sürelik genişletiyoruz. Aynı zamanda yankı ekliyoruz. İlk olarak döngüye alıyoruz ve sonsuza kadar sürmesini sağlıyoruz. Bu demek oluyor ki her döngüde yeni bir yankı yaratıyoruz. Bunu daha benzer yapmak için gitaristimize sadece bir yankı pedalı veriyoruz.

```
with_fx :reverb do
  loop do
        play 60, release: 0.1
        sleep 0.125
  end
end
```

Döngüyü içeriye with_fx bloğu kullanarak al bu sayede sadece bir yankı üreteceksin. Bu kod daha efektif. Hadi with_fx kullanalım

```
loop do
  with_fx :reverb do
        16.times do
        play 60, release: 0.1
        sleep 0.125
        end
  end
end
```

7.1 Oynatılan Synth leri kontrol etmek

Bu zamana kadar yeni sesler tetiklemeyi ve FX kullanımını öğrendik. Sonic Pi bize oynatılan sesler üzerinde kontrol hakkı tanıyor.

```
s = play 60, release: 5
```

burada bir değişkenimiz var s. Bu değişken 60. Notayı oynamamızı sağlıyor fakat bu değişkene fonksiyonlar gibi ulaşamazsın. Bunun control konfsiyonu ile yapman lazım

```
s = play 60, release: 5
sleep 0.5
```

```
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```
burada 4 farklı synth i tetiklemiyoruz, yaptığımız şey tek birini tetiklemektir. Ardından oynatılırken 3 kere değiştriyoruz.

7.2 FX kontrolü

FX leri kontrol etmekte mümkündür fakat bunun yolu biraz daha farklı.

```
with_fx :reverb do |r|
  play 50
  sleep 0.5
  control r, mix: 0.7
  play 55
  sleep 1
  control r, mix: 0.9
  sleep 1
  play 62
end
```
bir değişken kullanak yerine | | kullanımı görüyoruz bunu do end bloğunda kullanmamız gerekiyor. Şimdi git biraz FX kontrol et.

7.3 Kayan Opts

synth leri ve FX opts ları keşfediyoruz. Fark ettiysen bir sürü sonu _slide ile biten opts var. Belki bir kısmını çağırmayıda denemiş olabilirsin fakat bi sonuç göremedin. Bunun sebebi bunlar normal parametreler değil. Şu örneği incele.

```
s = play 60, release: 5
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```
Duyduğun üzere sesler her contol çağrısında anında değişiyor. Fakat belki bazı yerlerin değişimler sırasında kaymasını istiyor olabilirsin. Note: ları kontrol ettşpşmşz gibi note_slide ıda ayarlamamız gerekiyor:

```
s = play 60, release: 5, note_slide: 1
```

```
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

FX opts kaydırma

Aynı zamanda Fx opts kaydırmakta mümkün

```
with_fx :wobble, phase: 1, phase_slide: 5 do |e|
  use_synth :dsaw
  play 50, release: 5
  control e, phase: 0.025
end
```

8 Veri Yapıları

Bir programcının araç setinde çok faydalı bir araç bir veri yapısıdır.

Bazen birden fazla şeyi temsil etmek ve kullanmak isteyebilirsiniz. Örneğin, birbiri ardında çalmak için bir dizi nota almanız yararlı olabilir. Programlama dilleri tam olarak bunu yapmanıza izin veren veri yapılarına sahiptir.

Programcılara sunulan birçok heyecan verici ve egzotik veri yapısı var - ve insanlar her zaman yenilerini icat ediyorlar. Ancak, şimdilik sadece çok basit bir veri yapısını dikkate almamız gerekiyor - liste.

8.1 Listeler

Bu bölümde çok yararlı olan bir veri yapısına bakacağız - liste. Notaları çalmak için bir not listesinden rastgele seçmek istediğimizde kullandığımız kod:

```
play choose([50, 55, 62])
```

Bu bölümde, akorları ve ölçekleri temsil etmek için listeleri kullanmayı da inceleyeceğiz. İlk önce nasıl bir akor çalabileceğimizi özetleyelim. Unutmayın ki sleep kullanmazsak, seslerin hepsi aynı anda olur:

```
play 52
play 55
play 59
```

Bu kodu temsil etmenin başka yollarına bakalım.

Liste Çalmak

Bir seçenek, tüm notları bir listeye yerleştirmektir: [52, 55, 59]. Dost canlısı play fonksiyonumuz notaların nasıl çalınacağını bilecek kadar akıllıdır. Hadi deneyelim:

```
play [52, 55, 59]
```

Okumak çok güzel. Bir not listesini çalmak, parametrelerden herhangi birini normal olarak kullanmanıza engel olmaz:

```
play [52, 55, 59], amp: 0.3
```

Elbette, MIDI numaraları yerine geleneksel not adlarını da kullanabilirsiniz:

```
play [:E3, :G3, :B3]
```

Şimdi sizler müzik teorisi çalıştıracak kadar şanslıysanız, 3. oktavda Mi-Minor çaldığı akoru tanıyabilir.

Listeye Ulaşmak

Bir listenin çok yararlı bir başka özelliği de ondan bilgi alma yeteneğidir. Bu biraz garip gelebilir, ancak sizden bir kitabı sayfa 23'e çevirmesini isteyen birinden daha karmaşık değildir. Listede,

23. indeks öğenin ne olduğunu söyleyeceksiniz? Tuhaf olan tek şey, programlama endekslerinde genellikle 1 değil 0'da başlamasıdır.

Liste dizinleri (indeks) ile 1, 2, 3 sayılmaz… Bunun yerine 0, 1, 2 olarak sayıyoruz…

Buna biraz daha ayrıntılı bakalım. Bu listeye bir göz atın:

```
[52, 55, 59]
```

Bu konuda özellikle korkutucu bir şey yok. Şimdi, bu listedeki ikinci unsur nedir? Evet, tabii ki, 55. Bu kolaydı. Bakalım, bilgisayarı bizim için de cevaplayabiliyor mu?

```
puts [52, 55, 59][1]
```

Tamam, daha önce hiç böyle bir şey görmediyseniz bu biraz garip görünüyor. İnan bana, çok zor değil. Yukarıdaki satırda üç bölüm vardı: kelime puts, listemiz 52, 55, 59 ve indeksimiz [1]. İlk önce puts kullanıyoruz, çünkü Sonic Pi'nin cevabı bizim için sistemde yazdırmasını istiyoruz. Daha sonra, listemizi veriyoruz ve nihayet endeksimiz ikinci elemanı istiyor. Endeksimizi köşeli parantezlerle sarmamız gerekiyor ve sayım 0'dan başladığından, ikinci indeks 1'dir.

```
# indexes:  0   1   2
           [52, 55, 59]
```

8.2 Akorlar

Sonic Pi, listeleri döndürecek akor isimleri için dahili desteğe sahiptir. Siz de deneyin:

```
play chord(:E3, :minor)
```

Şimdi, gerçekten bir yerlere ulaşıyoruz. Bu ham listelerden çok daha hoş görünüyor (ve diğer insanlar için okunması daha kolay). Peki Sonic Pi başka hangi akorları destekliyor? Bunlardan bazılarını deneyin:

- `chord(:E3, :m7)`
- `chord(:E3, :minor)`
- `chord(:E3, :dim7)`
- `chord(:E3, :dom7)`

Arpejler

`Play_pattern`: fonksiyonu ile kolayca akorları arpejlere çevirebiliriz.

```
play_pattern chord(:E3, :m7)
```

play_pattern listedeki her notayı çalacak bir çağrı ile ayrılmış olarak çalacak her çağrı arasında sleep 1 ile çalacak. Kendi zamanlamamızı belirlemek ve işleri hızlandırmak için play_pattern_timed fonksiyonunu kullanabiliriz:

```
play_pattern_timed chord(:E3, :m7), 0.25
```

You can also use this:

```
play_pattern_timed chord(:E3, :m13), [0.25, 0.5]
```

Bu şuna eşdeğerdir:

```
play 52
sleep 0.25
```

```
play 55
sleep 0.5
play 59
sleep 0.25
play 62
sleep 0.5
play 66
sleep 0.25
play 69
sleep 0.5
play 73
```

8.3 Ölçekler

Sonic Pi, çok çeşitli ölçekleri desteklemektedir. Bir C3 major ölçekli oynamaya ne dersiniz?

```
play_pattern_timed scale(:c3, :major), 0.125, release: 0.1
```

Daha fazla oktav bile isteyebiliriz:

```
play_pattern_timed scale(:c3, :major, num_octaves: 3), 0.125,
release: 0.1
```

How about all the notes in a pentatonic scale?

```
play_pattern_timed (scale :c3, :major_pentatonic, num_octaves:
3), 0.125,

  release: 0.1
```

Rastgele Notalar

Akorlar ve ölçekler, rastgele bir seçimi anlamlı bir şeyle sınırlamanın harika yollarıdır. Akor E3 minöründen rastgele notaları seçen bu örnek ile oynayalım:

```
use_synth :tb303
loop do
  play choose(chord(:E3, :minor)), release: 0.3, cutoff:
rrand(60, 120)
  sleep 0.25
end
```

8.4 Halkalar

Listelerin önceki bölümünde, indeksleme mekanizmasını kullanarak elementleri onlardan nasıl alabileceğimizi gördük:

```
puts [52, 55, 59][1]
```

Şimdi, endeks 100'ü istersen ne olur? Listede sadece üç unsur bulunduğundan, endeks 100'de hiçbir eleman yoktur. Sonic Pi size nil getirecek, yani hiçbir şey ifade etmiyor.

Bununla birlikte, sürekli olarak artan mevcut ritim gibi bir sayacınız olduğunu düşünün. Sayacımızı ve listemizi oluşturalım:

```
counter = 0

notes = [52, 55, 59]
```

Artık listemizdeki bir nota erişmek için sayacımızı kullanabiliriz:

```
puts notes[counter]
```

Harika, 52 oldu. Şimdi, sayacımızı yükseltelim ve başka bir not alalım:

```
counter = (inc counter)

puts notes[counter]
```

Süper, şimdi 55 elde ediyoruz ve tekrar yaparsak 59 elde ediyoruz. Ancak, tekrar yaparsak listemizdeki sayıları tükenir ve nil alırız. Ya sadece geri dönüp tekrar listenin başında başlamak istersek? Halkalar bunun için var.

Halka Yaratmak

İki yoldan biriyle halkalar oluşturabiliriz. Ya halka işlevini halkanın elemanları ile birlikte parametre olarak kullanırız:

```
(ring 52, 55, 59)
```

Veya normal bir listeyi alıp .ring mesajını göndererek bir zil sesi haline getirebiliriz:

```
[52, 55, 59].ring
```

Halkaları indekslemek

Bir halkamız olduğunda, halkanın boyutundan negatif veya daha büyük olan indeksleri kullanabilmeniz ve normalde çembere saracak olması dışında normal bir listeyi kullandığınız gibi kullanabilirsiniz. Halkanın elementlerinden biri:

```
(ring 52, 55, 59)[0] #=> 52

(ring 52, 55, 59)[1] #=> 55
```

```
(ring 52, 55, 59)[2] #=> 59

(ring 52, 55, 59)[3] #=> 52

(ring 52, 55, 59)[-1] #=> 59
```

8.5 - Halka Zincirleri

range ve yeni halkalar yaratmanın başka bir yolu spread gibi “yapıcılara” ek olarak mevcut halkaları manipüle etmektir.

Zincir Komutları

Örnek halka;

```
(ring 10, 20, 30, 40, 50)
```

Ya geri istiyorsak? Zincir komutu .reverse kullanırız. Yüzüğü alıp geri almak için tersine çevirin:

```
(ring 10, 20, 30, 40, 50).reverse  #=> (ring 50, 40, 30, 20, 10)
```

Şimdi, halkadaki ilk üç değeri istersek?

```
(ring 10, 20, 30, 40, 50).take(3)  #=> (ring 10, 20, 30)
```

Son olarak, ya yüzüğü karıştırmak istiyorsak?

```
(ring 10, 20, 30, 40, 50).shuffle  #=> (ring 40, 30, 10, 50, 20)
```

Birden Çok Zincir

Bu zaten yeni yüzükler yaratmanın güçlü bir yoludur. Ancak, real güç, bu komutlardan birkaçını bir araya getirdiğimizde gelir.

Yüzüğü karıştırmaya, 1 elemanı düşürmeye ve sonra bir sonraki 3'ü almaya ne dersiniz?

1. `(ring 10, 20, 30, 40, 50)` - ilk halkamız

2. `(ring 10, 20, 30, 40, 50).shuffle` - karıştırır - `(ring 40, 30, 10, 50, 20)`

3. `(ring 10, 20, 30, 40, 50).shuffle.drop(1)` - 1'i düşürme - `(ring 30, 10, 50, 20)`

4. `(ring 10, 20, 30, 40, 50).shuffle.drop(1).take(3)` - 3'ü alma - `(ring 30, 10, 50)`

Bunları bir araya getirerek nasıl bu yöntemlerin uzun bir zincirini oluşturabileceğimizi görebiliyor musunuz? Bunları var olanlardan yeni halkalar üretmenin son derece zengin ve güçlü bir yolunu oluşturmak istediğimiz herhangi bir sırayla birleştirebiliriz.

Mevcut Zincir Yöntemleri

`.reverse` - halkanın tersine çevrilmiş halini döndürür
`.sort` - halkanın sıralanmış bir versiyonunu oluşturur
`.shuffle` - halkanın karıştırılmış bir versiyonunu oluşturur
`.pick(3)` - 3 kez .choose çağıran sonuçları gösteren bir yüzük döndürür
`.pick` - .pick'e (3) benzer, yalnızca boyut varsayılan olarak orijinal zil sesiyle aynıdır
`.take(5)` - sadece ilk 5 öğeyi içeren yeni bir yüzük döndürür
`.drop(3)` - ilk 3 öğeden başka her şeyde yeni bir yüzük döndürür
`.butlast` - son eleman eksikken yeni bir yüzük döndürür

`.drop_last(3)` - son 3 element eksik olan yeni bir yüzük döndürür
`.take_last(6)` - sadece son 6 element ile yeni bir yüzük döndürür
`.stretch(2)` - halkadaki her bir öğeyi iki kez tekrarlar
`.repeat(3)` - tüm halkayı 3 kez tekrarlar
`.mirror` - yüzüğü tersine çevrilmiş bir sürüme ekler
`.reflect` - ayna ile aynıdır, ancak orta değeri kopyalanamaz
`.scale(2)` - tüm elemanların 2 ile çarpılmasıyla yeni bir zil sesi döndürür (halkanın sadece sayılar içerdiğini varsayar)

9.1 - Canlı Kodlama

Şimdi gerçekten eğlenmeye başlamak için yeterince şey öğrendik. Bu bölümde, önceki bölümlerin tümünden çizim yapacağız ve size müzik bestelerinizi nasıl canlı hale getirip performansa dönüştürebileceğinizi göstereceğiz. Bunun için 3 ana malzemeye ihtiyacımız olacak:

-Ses çıkaran kod yazma yeteneği

-İşlev yazma yeteneği

-(Adlandırılmış) konuları kullanma yeteneği

Haydi ilk sesimizi kodlayalım. Önce oynamak istediğimiz kodu içeren bir işleve ihtiyacımız var. Basit başlayalım. Ayrıca bir iş parçacığında çağrıları bu işleve ilmek istiyoruz:

```
define :my_sound do
```

```
  play 50
  sleep 1
end


in_thread(name: :looper) do
  loop do
    my_sound
  end
end
```

Bu size biraz karmaşık geliyorsa, geri dönün ve işlevler ve ipliklerdeki bölümleri tekrar okuyun. Bu gibi şeylere kafanızı sardıysanız, bu çok karmaşık değildir.

Burada sahip olduğumuz şey, sadece 50 no'lu notaları çalan ve bir vuruş için uyuyan bir fonksiyon tanımıdır. Daha sonra sadece my_sound'u tekrar tekrar çağırmak için dolaşan :looper adında bir konu tanımladık.

Bu kodu çalıştırırsanız, not 50'nin tekrar tekrar tekrar ettiğini duyacaksınız…

Değiştirmek

Kod hala 50 iken başka bir numaraya değiştirilirken, 55 deyin, ardından tekrar Çalıştır düğmesine basın.

Yeni bir katman eklemedi, çünkü her ad için yalnızca bir iş parçacığına izin veren bir iş parçacığı kullanıyoruz. Ayrıca, işlevi değiştirdik çünkü ses değişti. Verdik :my_sound yeni bir tanım. :looper ipliği etrafında dolandığında basitçe yeni tanım denir.

Tekrar değiştirmeyi deneyin, notu değiştirin, uyku süresini değiştirin. use_synth ifadesi eklemeye ne dersiniz? Örneğin, şuna değiştirin:

```
define :my_sound do

  use_synth :tb303

  play 50, release: 0.3

  sleep 0.25

end
```

Aynı notayı tekrar tekrar çalmak yerine, bir akor çalmayı deneyin:

```
define :my_sound do

  use_synth :tb303

  play chord(:e3, :minor), release: 0.3

  sleep 0.5

end
```

Akordan rastgele nota çalmaya ne dersiniz:

```
define :my_sound do

  use_synth :tb303

  play choose(chord(:e3, :minor)), release: 0.3

  sleep 0.25

end
```

Veya rastgele bir kesme değeri kullanarak:

```
define :my_sound do
```

```
  use_synth :tb303

  play choose(chord(:e3, :minor)), release: 0.2, cutoff:
rrand(60, 130)

  sleep 0.25

end
```

Son olarak, bazı davul ekleyin:

```
define :my_sound do

  use_synth :tb303

  sample :drum_bass_hard, rate: rrand(0.5, 2)

  play choose(chord(:e3, :minor)), release: 0.2, cutoff:
rrand(60, 130)

  sleep 0.25

end
```

9.2 - Canlı Döngüler

Öğreticinin bu bölümü gerçek bir mücevher. Sadece bir bölümü okursanız, o bu olmalı. Canlı Kodlama Temelleri ile ilgili önceki bölümü okursanız, live_loop tam olarak bunu yapmanın basit bir yoludur, ancak çok fazla yazı yazmak zorunda kalmadan.

Önceki bölümü okumadıysanız, live_loop Sonic Pi ile uğraşmanın en iyi yoludur.

Hadi oynayalım. Aşağıdakileri yeni bir belleğe yazın:

```
live_loop :foo do
  play 60
  sleep 1
end
```

Şimdi Çalıştır düğmesine basın. Her vuruşta temel bir bip sesi duyarsınız. Orada eğlenceli bir şey yok. Ancak, henüz Durdur düğmesine basmayın. 60 ila 65'i değiştirin ve tekrar Çalıştır'a basın.

Vay! Bir ritmi kaçırmadan otomatik olarak değişti. Bu canlı kodlama.

Neden onu daha çok bas gibi değiştirmiyorsun? Oynarken kodunuzu güncellemeniz yeterlidir:

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8
```

```
  sleep 8
end
```

Ardından Çalıştır'a basın.

Kesmeyi hareket ettirelim:

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end
```

Tekrar Çalıştır'a basın.

Bazı davul ekleyin:

```
live_loop :foo do
  sample :loop_garzul
  use_synth :prophet
  play :e1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end
```

Notu e1'den c1'e değiştirin:

```
live_loop :foo do

  sample :loop_garzul

  use_synth :prophet

  play :c1, release: 8, cutoff: rrand(70, 130)

  sleep 8

end
```

9.3 - Birden çok Canlı Döngüler

Aşağıdaki canlı döngüyü düşünün:

```
live_loop :foo do
  play 50
  sleep 1
end
```

Neden bu isme ihtiyaç duyduğunu merak etmiş olabilirsin :foo. Bu ad önemlidir, çünkü bu canlı döngünün diğer tüm canlı döngülerden farklı olduğunu gösterir.

Aynı adla çalışan iki canlı döngü asla olamaz.

Bu, aynı anda birden fazla çalışan canlı döngü istiyorsak, onlara farklı isimler vermemiz gerektiği anlamına gelir:

```
live_loop :foo do
  use_synth :prophet
  play :c1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end

live_loop :bar do
  sample :bd_haus
  sleep 0.5
end
```

Canlı Döngüleri Senkronize Etme

Önceden fark etmiş olabileceğiniz bir şey, canlı döngülerin daha önce araştırdığımız iş parçacığı işaret mekanizmasıyla otomatik olarak çalışmasıdır. Canlı döngü her döngüde, canlı döngü adıyla yeni bir işaret olayı oluşturur. Bu nedenle, döngülerimizin hiçbir şeyi durdurmak zorunda kalmadan senkronize olmalarını sağlamak için bu ipuçlarını senkronize edebiliriz.

Bu kötü senkronize edilmiş kodu düşünün:

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.4
end

live_loop :bar do
  sample :bd_haus
  sleep 1
end
```

Bakalım, zamanlamayı ve senkronizasyonu durdurmadan düzeltebilir miyiz. İlk önce, uykuyu 1 faktörü yapmak için :foo döngüsünü düzeltelim: 0,5 gibi bir şey:

```
live_loop :foo do
```

```
  play :e4, release: 0.5

  sleep 0.5

end



live_loop :bar do

  sample :bd_haus

  sleep 1

end
```

Henüz tam olarak bitmedik - atımların tam olarak hizalanmadığını fark edeceksiniz. Bunun nedeni, halkaların aşama dışı kalmasıdır. Birini diğerine senkronize ederek bunu düzeltelim:

```
live_loop :foo do

  play :e4, release: 0.5

  sleep 0.5

end
```

```
live_loop :bar do
  sync :foo
  sample :bd_haus
  sleep 1
end
```

10 - Time State

Genellikle, birden çok iş parçacığı veya canlı döngüde paylaşılan bilgilerin olması yararlıdır. Örneğin, mevcut anahtar, BPM veya mevcut 'karmaşıklık' (hatta farklı konular arasında farklı şekillerde potansiyel olarak yorumlayacağınız) gibi daha soyut kavramları paylaşabilirsiniz. Ayrıca, bunu yaparken mevcut determinizm garantilerimizi kaybetmek istemiyoruz. Başka bir deyişle, kodu başkalarıyla paylaşmayı ve çalıştırırken ne duyacaklarını tam olarak bilmek istiyoruz. Bu eğitimin 5.6. Bölümünün sonunda,

determinizm kaybı nedeniyle (yarış koşulları nedeniyle), konuları paylaşmak için değişkenleri neden kullanmamamız gerektiğine kısaca değindik.

Sonic Pi'nin, küresel değişkenlerle belirleyici bir şekilde kolayca çalışabilmesi sorununa çözümü, Time State adını verdiği yeni bir sistemdir. Bu kulağa karmaşık ve zor gelebilir (aslında, İngiltere'de birden fazla iş parçacığı ve paylaşılan hafıza ile programlama tipik olarak üniversite düzeyinde bir konudur). Ancak, göreceğiniz gibi, tıpkı ilk notunuzu çalmak gibi, Sonic Pi, programlarınızı iş parçacığı için güvenli ve deterministik tutarken hala konuları paylaşmayı inanılmaz derecede kolaylaştırıyor.

get ve set ile tanışın.

10.1 - Get ve Set

Set

Bilgileri Zaman durumu kaydetmek için iki şeye ihtiyacımız var:

1.saklamak istediğimiz bilgileri,

2.bilgi için benzersiz bir ad (anahtar).

Örneğin, 3000 sayısını şu anahtarla depolamak isteyebiliriz :intensity. Bu, set işlevini kullanarak mümkündür:

```
set :intensity, 3000
```

Anahtarımız için herhangi bir ismi kullanabiliriz. Bilgi bu anahtarla önceden kaydedilmişse, yeni setimiz geçersiz kılar:

```
set :intensity, 1000
```

```
set :intensity, 3000
```

Yukarıdaki örnekte, her iki sayıyı da aynı anahtarın altına koyduğumuzda, set "kazanır" olarak yapılan son çağrı, yani: ile ilişkili sayı :intensity, set için yapılan ilk çağrı etkin olarak geçersiz kılındığından 3000 olacaktır.

Get

Zaman Durumundan bilgi almak için, sadece bizim durumumuzda :intensity set için kullandığımız anahtara ihtiyacımız var. O zaman sadece sonucu sisteme yazdırarak görebileceğimiz get [: intensity] olarak çağırmamız gerekir:

```
print get[:intensity] #=> prints 3000
```

Birden çok Konu

```
live_loop :setter do
  set :foo, rrand(70, 130)
  sleep 1
end

live_loop :getter do
  puts get[:foo]
  sleep 0.5
end
```

get ve set konularını bu şekilde ayarlamak ile ilgili güzel şey, kodu oynatmaya başladığınızda her zaman aynı sonucu vermesidir. Hadi, dene. Sisteminin aşağıdakileri alıp almadığını gör.

```
{run: 0, time: 0.0}

 └─ 125.72265625
```

```
{run: 0, time: 0.5}

 └─ 125.72265625
```

```
{run: 0, time: 1.0}
```

└ 76.26220703125

{run: 0, time: 1.5}

└ 76.26220703125

{run: 0, time: 2.0}

└ 114.93408203125

{run: 0, time: 2.5}

└ 114.93408203125

{run: 0, time: 3.0}

└ 75.6048583984375

```
{run: 0, time: 3.5}

 └─ 75.6048583984375
```

Basit Deterministik Durum Sistemi

Bölüm 5.6'ya geri dönersek, değişkenler arasındaki değişkenleri kullanmanın neden rastgele davranışa neden olabileceğini tartıştık. Bu, bunun gibi bir kodu güvenilir bir şekilde yeniden üretmemizi engeller:

```
a = (ring 6, 5, 4, 3, 2, 1)

live_loop :shuffled do
  a = a.shuffle
  sleep 0.5
end

live_loop :sorted do
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
End

------------------------------------------------------------

set :a, (ring 6, 5, 4, 3, 2, 1)



live_loop :shuffled do
```

```
  set :a, get[:a].shuffle
  sleep 0.5
end

live_loop :sorted do
  set :a, get[:a].sort
  sleep 0.5
  puts "sorted: ", get[:a]
End
```

10.2 - Senkronize (Sync)

Etkinlikleri Bekleme

Zaman Durumuna yeni etkinliklerin eklenmesini beklemek için sync nasıl kullanılacağına hızlıca göz atalım:

```
in_thread do
  sync :foo
  sample :ambi_lunar_land
end

sleep 2

set :foo, 1
```

Bu örnekte, önce Zaman Durumuna eklenecek bir :foo olayını bekleyen bir iş parçacığı oluşturuyoruz. Bu iş parçacığı bildirimi sonra 2 vuruş için uyur ve sonra set :foo 1 olur. Bu daha sonra bir sonraki satıra hareket eden sync serbest bırakır :ambi_lunar_land (örnek olarak).

Değerleri Geleceğe Geçirmek

Yukarıdaki örnekte belirledik :foo 1 ile hiçbir şey yapmadık. Bu değeri thread thread sync'den gerçekten alabiliriz:

```
in_thread do
  amp = sync :foo
  sample :ambi_lunar_land, amp: amp
end

sleep 2

set :foo, 0.5
```

10.3 - Desen Eşleştirme

Zaman Durumuna bilgi alırken ve ayarlarken, :foo ve :bar gibi temel sembollerden daha karmaşık anahtarlar kullanmak mümkündür. "/ Foo / bar / baz" gibi yollar adı verilen URL stil dizelerini de kullanabilirsiniz. Yollarla çalışmaya başladığımızda, Sonic Pi'nin 'özdeş' yollarından ziyade 'benzer' ile sync ve get kullanmak için sofistike kalıp eşleştirme sisteminden yararlanmaya başlayabiliriz.

11 - MIDI

Bir kez kodu müziğe dönüştürme ustalaşmayı merak edebilirsin - sırada ne var? Bazen sadece Sonic Pi'nin sözdizimi ve ses sistemi içerisinde çalışmanın kısıtları heyecan verici olabilir ve sizi yeni bir yaratıcı pozisyona getirebilir. Bununla birlikte, bazen kodu gerçek dünyaya atmak çok önemlidir. İki ekstra şey istiyoruz:

1. Gerçek dünyadaki eylemleri Sonic Pi olaylarına dönüştürebilmek
2. Gerçek dünyadaki nesneleri kontrol etmek ve değiştirmek için Sonic Pi'nin güçlü zamanlama modelini ve anlam bilimini kullanabilmek

11.1 - MIDI Girişi

```
live_loop :midi_piano do
  note, velocity = sync "/midi/nanokey2_keyboard/0/1/note_on"
  synth :piano, note: note
End
```

11.1 - MIDI Çıkışı

MIDI olaylarını almaya ek olarak, harici donanım sentezlerini, klavyeleri ve diğer cihazları tetiklemek ve kontrol etmek için MIDI olayları gönderebiliriz. Sonic Pi, aşağıdakiler gibi çeşitli MIDI mesajlarını göndermek için tam bir set sunar:

1. Not açık - midi_note_on
2. Not kapalı - midi_note_off
3. Kontrol değişikliği - midi cc
4. Saat sesleri - midi clock_tick

Örnek:

```
with_fx :reverb, room: 1 do
  live_audio :moog
end

live_loop :moog_trigger do
  use_real_time
  midi (octs :e1, 3).tick, sustain: 0.1
  sleep 0.125
end
```

12 - OSC

MIDI'ye ek olarak, Sonic Pi'ye girip bilgi almanın başka bir yolu da, OSC - Açık Ses Kontrolü adı verilen basit bir protokol kullanarak ağ üzerinden yapmaktır. Bu, 1980'lerin tasarımı nedeniyle sınırlamaları olan MIDI'nın ötesinde kontrol potansiyeli açan harici programlara (hem bilgisayarınızda hem de harici bilgisayarlarda çalışan) mesaj göndermenizi sağlar.

12.1 - OSC’yi Anlamak

Varsayılan olarak, Sonic Pi başlatıldığında, aynı bilgisayardaki programlardan gelen OSC mesajları için 4559 numaralı bağlantı noktasını dinler. Bu, herhangi bir konfigürasyon olmadan Sonic Pi'ye bir OSC mesajı gönderebileceğiniz ve gelen MIDI mesajlarında olduğu gibi işaret günlüğünde görüntüleneceği anlamına gelir. Bu aynı zamanda, gelen herhangi bir OSC mesajının da Zaman Durumuna otomatik olarak ekleneceği anlamına gelir; bu, aynı zamanda MIDI ve aynı şekilde live_loops senkronizasyonu gibi gelen verilerle çalışmak için get ve sync'i kullanabileceğiniz anlamına gelir - bunun nasıl çalıştığını özetlemek için 5.7 ve 10.2 bölümlerine bakın.

```
live_loop :foo do
  use_real_time
  a, b, c = sync "/osc/trigger/prophet"
  synth :prophet, note: a, cutoff: b, sustain: c
end
```

12.2 - OSC Yollamak

```
use_osc "localhost", 4559
osc "/hello/world"
```

Yukarıdaki kodu çalıştırırsanız, Sonic Pi'nin kendisine bir OSC mesajı gönderdiğini fark edeceksiniz! Bunun sebebi IP adresini mevcut makineye ve portu porttaki varsayılan OSC'ye ayarlamamızdır. Bu aslında kendinize bir mektup göndermekle aynıdır - OSC paketi yaratılır, Sonic Pi'den çıkar, işletim sisteminin ağ yığınına girer, ardından paketlenmiş olanı Sonic Pi'ye yönlendirir ve daha sonra standart bir OSC mesajı olarak alınır ve cue logger'da gelen mesaj olarak görünür "/ osc / hello / world". (Sonic Pi'nin gelen tüm OSC mesajlarının /osc ile otomatik olarak nasıl öneklendiğine dikkat edin.)

13 - Çok Kanallı Ses

Şimdiye kadar, ses üretimi açısından, play, synth ve sample üzerinden senkronizeleri ve kaydedilmiş sesleri tetiklemeyi araştırdık. Bunlar daha sonra stereo hoparlör sistemimizde çalınan sesi üretti. Bununla birlikte, birçok bilgisayar, ikiden fazla hoparlöre ses gönderme özelliğine ek olarak, belki bir mikrofon aracılığıyla ses girişi de

yapabilir. Çoğu zaman, bu özellik harici bir ses kartı kullanılarak mümkün olur - bunlar tüm platformlarda kullanılabilir. Eğitimin bu bölümünde, bu harici ses kartlarından nasıl yararlanabileceğimize ve Sonic Pi'nin içinde ve dışında çok sayıda ses kanalıyla zahmetsizce çalışabileceğimize bir göz atacağız.

13.1 - Ses Girişi (Sound In)

Ses girişlerine erişmenin basit (ve belki de bilinen) bir yolu, synth kullanarak aşağıdakileri belirterek :sound_in

```
synth :sound_in
```

13.2 - Canlı Ses (Live Audio)

Önceki bölümde açıklandığı gibi: sound_in synth, giriş sesiyle çalışmak için çok esnek ve tanıdık bir yöntem sağlar. Bununla birlikte, tartışıldığı gibi, tek bir enstrüman olarak tek bir ses girişi ile çalışırken (ses veya gitar gibi) birkaç sorun vardır. Tek bir sürekli ses akışıyla çalışmak için en iyi yaklaşım live_audio kullanmaktır.

```
live_audio :foo

live_audio :foo, :stop

live_audio :foo, stereo: true, input: 2
```

13.3 - Ses Çıkışı (Sound Out)

Şimdiye kadar bu bölümde, Sonic Pi'ye birden fazla ses akışının nasıl alınacağına - :sound_in synth ya da güçlü live_audio sistemi aracılığıyla baktık. Çoklu giriş ses akışlarıyla çalışmaya ek olarak, Sonic Pi çoklu ses akışlarını da verebilir. Bu şu şekilde elde edilir ---> :sound_out FX.

```
with_fx :reverb do       # C
  with_fx :sound_out, output: 3 do # B
    sample :bd_haus      # A
  end
end

with_fx :sound_out, output: 3, amp: 0 do # B
  sample :loop_amen                      # A
end
```