Temel Veri Yapıları

TODO: içindekiler veya bir tema resmi

2.1 Hedefler

- → Temel veri yapılarından *yığıt* (*stack*), *kuyruk* (*queue*) ve *deque* in mantıksal yapısını anlamak
- → Python'la bu ADT'leri (Soyut Veri Türleri) gerçeklemek
- → *ön-* (*prefix*), *iç-* (*infix*) ve *son-* (*postfix*) notasyonlu ifadeleri anlamak
- → *postfix* ifadelerini yığıtla değerlendirmek (hesaplamak)
- → *infix* --> *postfix* ifade çevrimini yapmak (yığıtla)
- → temel zamanlama simülasyonlarında kuyruk kullanma
- → yığıt, kuyruk ve deque'in hangi durumlarda uygun veri yapısı olduğunu tanıyabilmek

2.2 Doğrusal Yapılar Nedir?

Doğrusal Veri Yapısı yenisi ekleneceğinde, önüne veya arkasına diye yer tanımlayabildiğimiz koleksiyonlar

- → yığıt, kuyruk, deque
- → **doğrusal yapılar iki uçludur:**
  - → "sol" - "sağ"
  - → "ön" - "arka"
  - → "üst" - "alt"
- → bunlar uç isimleridir, daha ötesi değil!
- → hangi ismi verdiğinizin bir önemi yok
- → yeni öğeler bir (veya her iki) uçtan giriş yapar, diğer (veya her iki) uçtan çıkış yapar

2.3 Yığıtlar (Stack)

2.3.1 Yığıt Nedir?

- yığıt, "it-çek yığıt"
- ekleme / çıkarma aynı uçtan
- LIFO: son giren ilk çıkar
- yeni girenler üstte, eskiler altta (baza yakın)

Kitap *yığını*

- günlük hayatta tabak yığını
- kitap yığını olarak karşımıza çıkar
- tepede sadece bir kitap görünür, diğerlerine erişmek için üstünde olanların çıkarılması gerekir

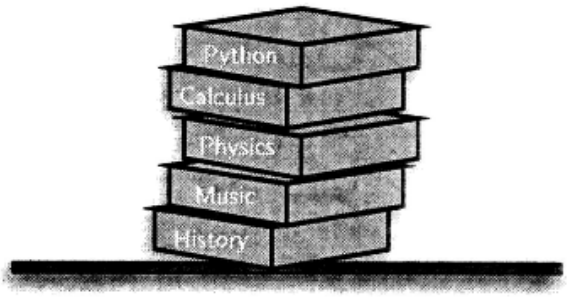


Figure 2.1: A Stack of Books

Python nesne yığını

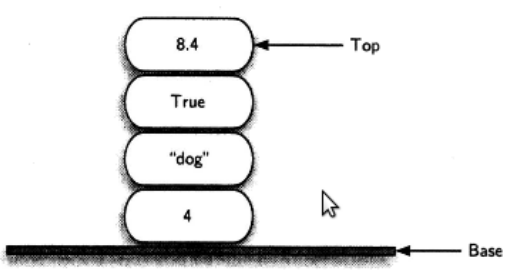


Figure 2.2: A Stack of Primitive Python Objects

Web tarayıcıları: ileri/geri tuşları

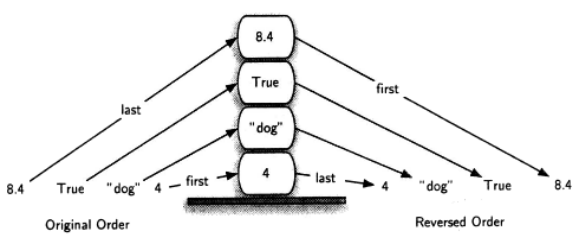


Figure 2.3: The Reversal Property of Stacks

2.3.2 Yığıt Soyut Veri Türü

Stack(): *boş, yeni bir yığıt oluşturur*

push(item): ***item*** *i (öğeyi) yığıtın tepesine ekler. Dönüş yok.*

pop(): *tepeden bir öğeyi çıkarır, geri döndürür. Yığıt güncellenir.*

peek(): *tepedeki değer döndürülür, çıkarılmaz. Yığıt güncellenmez.*

isEmpty(): *yığıt boş mu? Mantıksal değer döner.*

size(): *yığıtta kaç eleman var. Tamsayı değer döner.*

Örnek yığıt işlemleri

Stack Operation	Stack Contents	Return Value
s.isEmpty()	[]	True
s.push(4)	[4]	
s.push('dog')	[4,'dog']	
s.peek()	[4,'dog']	'dog'
s.push(True)	[4,'dog',True]	
s.size()	[4,'dog',True]	3
s.isEmpty()	[4,'dog',True]	False
s.push(8.4)	[4,'dog',True,8.4]	
s.pop()	[4,'dog',True]	8.4
s.pop()	[4,'dog']	True
s.size()	[4,'dog']	2

Table 2.1: Sample Stack Operations

TODO: Python da demo.

2.3.3 Python'da yığıtı gerçekleme

Sınıf: *Soyut Veri Türü gerçeklemede kullanılır*

Yöntem: *yığıtın işlevlerini gerçeklemede kullanılır*

Liste: *ADT için en uygun Python ilkel veri türü*

Tasarım kararları

→ listenin ucu, yığıtın tepesi mi? yoksa bazı mı?

→ append() - pop() X insert() - pop()

Listenin işlevleri

Method Name	Use	Explanation
append	alist.append(item)	Adds a new item to the end of a list
insert	alist.insert(i,item)	Inserts an item at the ith position in a list
pop	alist.pop()	Removes and returns the last item in a list
pop	alist.pop(i)	Removes and returns the ith item in a list
sort	alist.sort()	Modifies a list to be sorted
reverse	alist.reverse()	Modifies a list to be in reverse order
del	del alist[i]	Deletes the item in the ith position
index	alist.index(item)	Returns the index of the first occurrence of item
count	alist.count(item)	Returns the number of occurrences of item
remove	alist.remove(item)	Removes the first occurrence of item

Table 1.2: Methods Provided by Lists in Python

v1: Python'da yığıt gerçekleme (listenin sonu = yığıtın tepesi)

```
1       class Stack:
2           def __init__(self):
3               self.items = []
4
5           def isEmpty(self):
6               return self.items == []
7
8           def push(self, item):
9               self.items.append(item)
10
11          def pop(self):
12              return self.items.pop()
13
14          def peek(self):
15              return self.items[len(self.items)-1]
16
17          def size(self):
18              return len(self.items)
```


Test

```
>>> execfile("listing_2_1.py")
>>> s = Stack()
>>> s.isEmpty()
True
>>> s.push(4)
>>> s.push('dog')
>>> s.peek()
'dog'
>>> s.push(True)
>>> s.size()
3
>>> s.isEmpty()
False
>>> s.push(8.4)
>>> s.pop()
8.4000000000000004
>>> s.pop()
True
>>> s.size()
2
>>>
```


v2: Python'da yığıt gerçekleme (listenin başı = yığıtın tepesi)

```
1       class Stack:
2           def __init__(self):
3               self.items = []
4
5           def isEmpty(self):
6               return self.items == []
7
8           def push(self, item):
9               self.items.insert(0,item)
10
11          def pop(self):
12              return self.items.pop(0)
13
14          def peek(self):
15              return self.items[0]
16
17          def size(self):
18              return len(self.items)
```

2.3.4 Dengeli Parantez Problemi

→ Parantezler her yerde

→ Aritmetik işlemlerde: ((5 + 6) * (7 + 8)) / (4 + 3)

→ Lisp'de

```
(defun square(n)
    (* n n))
```


Dengeli Parantez

Dengeli Parantez her bir açma sembolü, doğru girintiyle kapama sembolüyle karşılanması durumu

→ Ör: dengeli parantez

```
( ( ) ( ) ( ) ( ) )
( ( ( ( ) ) ) )
( ( ) ( ( ( ) ) ( ) ) )
```

→ Ör: dengesiz parantez

```
( ( ( ( ( ( ) )
( ) ) )
( ( ) ( ) ( ( )
```


Problem tanımı

→ katarı soldan sağa tara

→ açma-kapama sembolleri dengeli mi?

Çözüme doğru: gözlem

→ en son açma parantezi, ilk (sonraki) kapama sembolüyle uyuşması gerekiyor

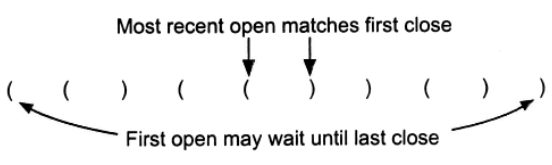


Figure 2.4: Matching Parentheses

Çözüme doğru: araç

- → kapama simgeleri ters sırada açma simgeleriyle eşleşir
- → Web tarayıcı ileri-geri düğmeleri
- → içten - dışa doğru eşleşme
- → en uygun araç: **yığıt**

Çözüm: Python - yığıt ile gerçekleme

```
1   def parChecker(symbolString):
2       s = Stack()
3
4       balanced = True
5       index = 0
6
7       while index < len(symbolString) and balanced:
8           symbol = symbolString[index]
9           if symbol == "(":
10              s.push(symbol)
11          else:
12              if s.isEmpty():
13                  balanced = False
14              else:
15                  s.pop()
16
17          index = index + 1
18
19      if balanced and s.isEmpty():
20          return True
21      else:
22          return False
```


Kod açıklama

- → boş yığıtla başla (s2)
- → soldan sağa katarı tara (s7-s8)
- → açma simgesini yığıta it (s9-s10)
- → kapama simgesine rastlayınca yığıttan çek (s15)
- → her kapama simgesi için yığıtta açma simgesi var mı? (s12)
- → yığıt boşken kapama simgesi geldiğinde (s13) veya
- → yığıt doluyken katarın sonuna ulaşıldığında (s22)
- → parantez katarı dengesizdir (s13, s22)
- → diğer durumda dengelidir (s20)

Geliştirme (TODO list)

- → kapama simgesi kontrolü yapılmıyor, herhangi bir karakter gözüyle bakılıyor (s11)

Test - demo

```
>>> parChecker('()')
True
>>> parChecker('())')
False
>>> parChecker('()()')
True
>>> parChecker('(()()')
False
>>> parChecker('(()())')
True
>>> parChecker('(()(())')
False
>>> parChecker('(()(()))')
True
>>>
```


2.3.5 Dengeli Simgeler (genelleştirme girişimi)

- → farklı türde açma - kapama simgesi vardır
- → **Python'da**
  - → listelerde - []
  - → sözlüklerde - {}
  - → tuple ve aritmek işlemlerde - ()
- → bunlar karışık bir şekilde bir arada olabilir
- → Ör: dengeli parantez

```
{ { ( [ ] [ ] ) } ( ) }
[ ] [ ] [ ] ( ) { }
```

- → Ör: dengesiz parantez

```
( [ ) ]
( ( ( ) ] ) )
[ { ( ) ]
```


Çözüm: Python - yığıt ile gerçekleme (genelleştirme girişimi)

```
1       def parChecker(symbolString):
2
3           s = Stack()
4
5           balanced = True
6           index = 0
7
8           while index < len(symbolString) and balanced:
9               symbol = symbolString[index]
10              if symbol in "([{":
11                  s.push(symbol)
12              else:
13                  if s.isEmpty():
14                      balanced = False
15                  else:
16                      top = s.pop()
17                      if not matches(top,symbol):
18                              balanced = False
19
20              index = index + 1
```


Çözüm: Python - yığıt ile gerçekleme (genelleştirme girişimi) (devam)

```
1           if balanced and s.isEmpty():
2               return True
3           else:
4               return False
5
6       def matches(open,close):
7           opens = "([{"
8           closers = ")]}"
9
10          return opens.index(open) == closers.index(close)
```

Kod açıklama

→ her bir açma simgesi, kendi **eşleniğiyle** sınanır (s17 ve s27-s31)

```
1 def matches(open,close):
2     opens = "([{"
3     closers = ")]}"
4
5     return opens.index(open) == closers.index(close)
```

→ iki simge uyuşmazsa dengesiz

→ katar taranır da, yığıt boşsa, yığıt dengeli (s22)

Test - demo

```
>>> execfile("listing_2_4.py")
>>> parChecker('()')
True
>>> parChecker('(}')
False
>>> parChecker('()}')
False
>>> parChecker('{()}')
True
>>> parChecker('[{()}')
False
>>> parChecker('[]{()}')
True
>>>
```


2.3.6 Onluk Sistemden İkilik Sisteme dönüşüm: dec2bin

→ bilgisayar ne bilir?

→ tamsayılar ise her yerde

→ ikisi arasında dönüşüm nasıl?

Onlu - ikili aritmetik

```
233_10 = 2 x 10^2 + 3 x 10^1 + 3 x 10^0
11101001_2 = 1 x 2^7 + 1 x 2^6 ...
```

Çözüme doğru: yöntem

- → dönüşüm algoritması "ikiye böl"
- → kalanı ters sırada birleştir
- → ters sırada - LIFO - Web sayfalarında ileri-geri tuşları
- → en uygun araç **yığıt**
- → kalanı yığıtta tut

Çözüme doğru: gözlem

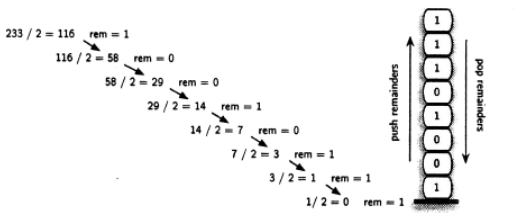


Figure 2.5: Decimal-to-Binary Conversion

Çözüm: Python - yığıt ile gerçekleme

```
1   def divideBy2(decNumber):
2
3       remstack = Stack()
4
5       while decNumber > 0:
6           rem = decNumber % 2
7           remstack.push(rem)
8           decNumber = decNumber / 2
9
10      binString = ""
11      while not remstack.isEmpty():
12          binString = binString + repr(remstack.pop())
13
14      return binString
```


Test - demo

```
>>> execfile("listing_2_5.py")
>>> divideBy2(20)
'10100'
>>> divideBy2(10)
'1010'
>>> divideBy2(7)
'111'
>>>
```

Geliştirme: herhangi bir tabanda dönüşüm

```
1   def baseConverter(decNumber,base):
2
3       digits = "0123456789ABCDEF"
4
5       remstack = Stack()
6
7       while decNumber > 0:
8           rem = decNumber % base
9           remstack.push(rem)
10          decNumber = decNumber / base
11
12      newString = ""
13      while not remstack.isEmpty():
14          newString = newString + digits[remstack.pop()]
15
16      return newString
```


Kod açıklama

→ kodda öne çıkan bölüm (s3, s13-s14)

```
1  digits = "0123456789ABCDEF"
2  ...
3  while not remstack.isEmpty():
4      newString = newString + digits[remstack.pop()]
```


Test - demo

```
>>> execfile("listing_2_6.py")
>>> baseConverter(23, 2)
'10111'
>>> baseConverter(23, 3)
'212'
>>> baseConverter(23, 8)
'27'
>>> baseConverter(23, 16)
'17'
>>> baseConverter(26, 16)
'1A'
>>>
```


2.3.7 Infix, Prefix ve Postfix Gösterimleri

B * C B ile C'yi çarp. * arada olunca **infix** notasyonu.

A + B * C İşlem önceliği. *, +'dan daha yüksek **önceliğe** sahiptir.

(A + B) * C **parantez** önceliği değiştirir.

A + B + C **soldan-sağa** kuralı.

Tam parantezli ifade

→ bilgisayar hangi işlemin, hangi sırada yapılacağını bilmek ister!

→ en kolay yol tüm işlemleri parantezle sarmalamak

→ **tam parantezli ifade**

```
A + B * C + D yerine     ( (A + (B * C) ) + D )
A + B + C + D yerine     ( ( (A + B) + C) + D )
```


Diğer gösterimler

→ işlecin nerede olduğuna bağlı olarak

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

Table 2.2: Examples of Infix, Prefix, and Postfix

Parantezsiz ifade

→ paranteze artık gerek

Infix Expression	Prefix Expression	Postfix Expression
(A + B) * C	* + A B C	A B + C *

Table 2.3: An Expression with Parentheses

Örnekler

Infix Expression	Prefix Expression	Postfix Expression
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Table 2.4: Additional Examples of Infix, Prefix, and Postfix

2.3.7.1 Infix-->Prefix ve Infix --> Postfix dönüşümü

→ postfix notasyonunda işleçleri kapama parantezine taşı

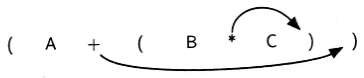


Figure 2.6: Moving Operators to the Right for Postfix Notation

Infix-->Prefix dönüşümü

→ işleçleri açma parantezine taşı

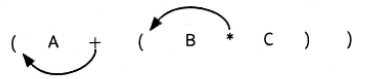


Figure 2.7: Moving Operators to the Left for Prefix Notation

Karmaşık ifadelerde dönüşüm

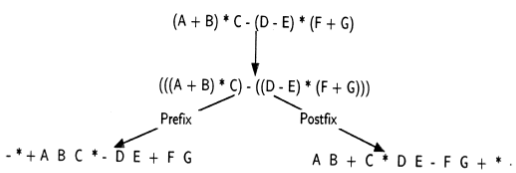


Figure 2.8: Converting a Complex Expression to Prefix and Postfix Notations

2.3.7.2 Genelleştirilmiş Infix-->Postfix Dönüşümü

→ Gözlem 1

`A + B * C --> A B C * +`

→ işlenenler göreceli olarak aynı konumda

→ işleçler yer değiştiriyor. **Neden?**

→ öncelik, sırayı değiştiren

Gözlem

`(A + B) * C --> A B + C *`

→ parantez en yüksek önceliğe sahiptir

→ sol parantez önceliği başlatır, sağ parantez bitirir

Tasarım ayrıntıları

→ Infix ifadeyi tara (soldan-sağa)

→ işleçler için yığıt

→ yığıtın tepesinde her zaman en son işleç

→ yeni işleç okunduğunda, işleç önceliğine bak

→ Infix ifadede ki katar parçaları: boşluk larla ayrılır

→ işleç parçaları ise: +-/* ve ( ).

→ işleçler tek karakterli.

Gerçekleme

```
1   import string
2   def infixToPostfix(infixexpr):
3
4       prec = {}
5       prec["*"] = 3
6       prec["/"] = 3
7       prec["+"] = 2
8       prec["-"] = 2
9       prec["("] = 1
10
11      opStack = Stack()
12      postfixList = []
13
14      tokenList = infixexpr.split()
15
16      for token in tokenList:
17          if token in string.uppercase:
18              postfixList.append(token)
19          elif token == '(':
20              opStack.push(token)
```


Gerçekleme (devam)

```
1           elif token == ')':
2               topToken = opStack.pop()
3               while topToken != '(':
4                   postfixList.append(topToken)
5                   topToken = opStack.pop()
6
7           else:
8               while (not opStack.isEmpty()) and \
9                  (prec[opStack.peek()] >= prec[token]):
10                   postfixList.append(opStack.pop())
11
12              opStack.push(token)
13
14      while not opStack.isEmpty():
15          postfixList.append(opStack.pop())
16
17      return string.join(postfixList)
```

Kod açıklama

1. İşleçleri tutacak opStack boş yığıtı oluştur (s11). Çıkış için boş liste oluştur (s12).
2. Giriş katarını split yöntemiyle listeye çevir (s14).

Kod açıklama (devam)

3. Soldan-sağa parça (**token**) listesini tara (s16).
   a. *Eğer işlenense, çıkış listesinin sonuna ekle (s17-s18).*
   b. *Sol parantezse,* opStack *e it (s19-s20).*
   c. *Sağ parantezse, eşi olan paranteze rastlayıncaya kadar* opStack*'ten çek (s21-s25). Her bir işleci çıkış listesinin sonuna ekle (s24).*
   d. *İşleçse,* opStack*'e it (s32). Fakat, ilk önce* opStack*'te olup daha yüksek veya eşit öncelikte olan işleçleri çek ve çıkış listesinin sonuna ekle (s28-s30).*
4. Giriş tamamen tarandığı halde opStack'te kalanları sırayla çek ve çıkış listesinin sonuna ekle (s34-s35).

Şematik gösterim

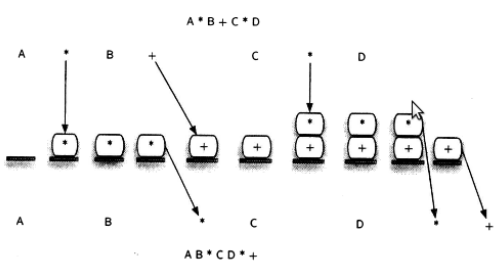
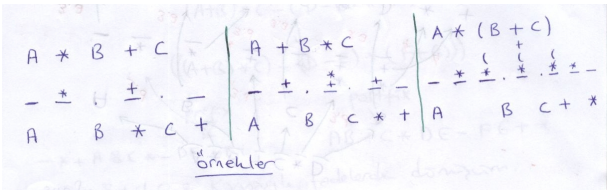


Figure 2.9: Converting A * B + C * D to Postfix Notation

Kod açıklama

→ prec sözlük veri türüne dikkat!

Örnek



Test - demo

```
>>> execfile("listing_2_7.py")
>>> infixToPostfix("A + B")
'A B +'
>>> infixToPostfix("A + B * C")
'A B C * +'
>>> infixToPostfix("( A + B ) * C")
'A B + C * '
>>> infixToPostfix("A + B  + C")
'A B + C +'
>>>
```


2.3.7.3 Postfix Değerlendirme

- → veri yapısı: yığıt
- → dönüştürmeden farkı yığıtta "işleçler" yerine "işlenenler" tutulur
- → girişçe "işleç"e rastlayınca son iki "işlenen" arasında "işlem" yap

Hesap sırasında yığıt içeriği

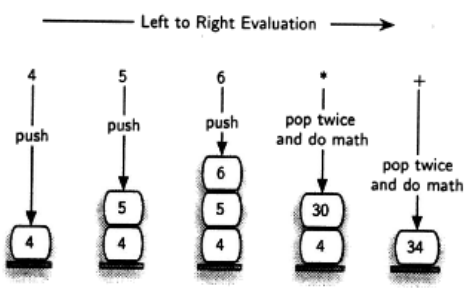


Figure 2.10: Stack Contents During Evaluation

Hesap sırasında yığıt içeriği: açıklama

- → 4 5 6 * + ardışıllığını ele alalım
- → 4 geldi ne yapacağız? (bilmiyorum, o zaman yığıta it),
- → 5 geldi ne yapacağız? (bilmiyorum, o zaman yığıta it),
- → 6 geldi ne yapacağız? (bilmiyorum, o zaman yığıta it),
- → * geldi ne yapacağız? (hıı, son iki işlenen üzerinde bu işlemi gerçekle)
- → yani: 5 * 6 => 30
- → 30'u ne yapayım? (bilmiyorum, o zaman yığıta it),
- → + geldi ne yapacağız? (hıı, son iki işlenen üzerinde bu işlemi gerçekle)
- → yani: 30 * 4 => 34
- → 34'ü ne yapayım? (bilmiyorum, o zaman yığıta it),
- → Katarın sonuna geldim ne yapayım? (yığıttaki değeri kullanıcıya söyle)

Daha karmaşık örnek

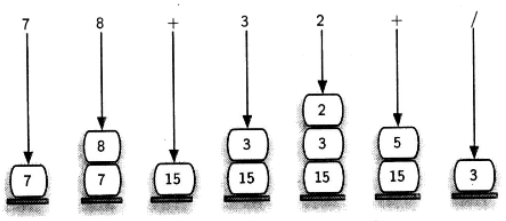


Figure 2.11: A More Complex Example of Evaluation

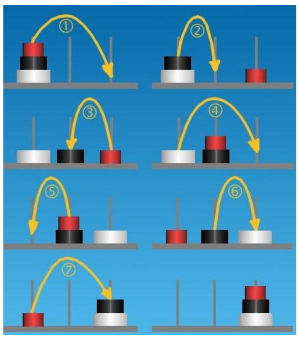
Kod açıklama

1. operandStack isimli boş bir yığıt oluştur
2. split() yöntemi yardımıyla katarı listeye dönüştür
3. Parça listesini soldan-sağa tara
   - → _işlenense_, tamsayıya çevir ve operandStack'e değer olarak it
   - → _işleçse_, iki işlenen gerektirir. operandStack'ten iki kez çek
   - → ilk çekilen ikinci işlenen, sonraki çekilen birinci işlenen
   - → Aritmetik işlemi gerçekleştir
   - → Sonucu operandStack'e it
4. Giriş ifadesi tamamen işlendiğinde, sonuç yığıttadır
   - → operandStack'ten çek ve değeri döndür


```
1   from listing_2_1 import Stack
2
3   def postfixEval(postfixExpr):
4
5       operandStack = Stack()
6
7       tokenList = postfixExpr.split()
8
9       for token in tokenList:
10          if token in "0123456789":
11              operandStack.push(int(token))
12          else:
13              operand2 = operandStack.pop()
14              operand1 = operandStack.pop()
15              result = doMath(token,operand1,operand2)
16              operandStack.push(result)
17
18      return operandStack.pop()
19
20  def doMath(op, op1, op2):
21      if op == "*":
22          return op1 * op2
23      else:
24          if op == "/":
25              return op1 / op2
26          else:
27              if op == "+":
28                  return op1 + op2
29              else:
30                  return op1 - op2
31
32  # Test
33  pstr = '2 4 3 * +'
34  result = postfixEval(pstr)
35  print pstr, " ==> ", result
```

Ödev 2

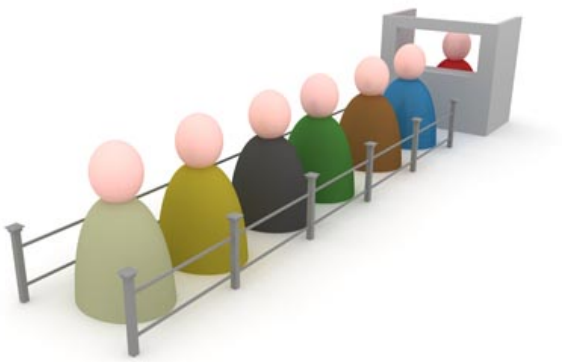
- → hanoi kuleleri
- → kurallar



- → demo: hanoi.py

2.4.1 Kuyruk nedir?

- → sıralı öğeler koleksiyonudur
- → doğrusal veri yapısıdır
- → yeni öğeler bir uca ("arkaya") eklenir, diğer uçtan ("ön") çıkarılır
- → FIFO, ilk gelene ilk servis yapılır



Örnek: gerçek hayattan

- → fatura, filim, market, kafeterya
- → kaynak yoktur, kuyruk iyi huyludur
- → ortaya atlama ve aradan sıyrılma yoktur

Python kuyruk

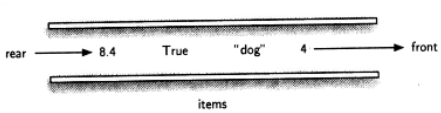


Figure 2.12: A Queue of Python Data Objects

Bilgisayar bilimlerinde

- → bilgisayar laboratuarındaki yazıcı
- → işletim sistemi, çoklu görevlidir
- → bilgisayarın klavyesi (kuyruklu tampon)

2.4.2 Kuyruk Soyut Veri Türü

Queue(): *boş kuyruk oluştur, parametre gerektirmez*

enqueue(item): *kuyruğun sonuna* item*'ı ekle*

dequeue(): *kuyruğun başındaki öğeyi çıkar, geri döndür, kuyruk güncellenir*

isEmpty(): *kuyruk boş mu? Mantıksal değer*

size(): *kuyrukta kaç öğe var? Tamsayı değer*

Temel işlemler

Queue Operation	Queue Contents	Return Value
q.isEmpty()	[]	True
q.enqueue(4)	[4]	
q.enqueue('dog')	['dog',4,]	
q.enqueue(True)	[True,'dog',4]	
q.size()	[True,'dog',4]	3
q.isEmpty()	[True,'dog',3]	False
q.enqueue(8.4)	[8.4,True,'dog',4]	
q.dequeue()	[8.4,True,'dog']	4
q.dequeue()	[8.4,True]	'dog'
q.size()	[8.4,True]	2

Table 2.5: Example Queue Operations

2.4.3 Python'da Kuyruk Gerçekleme

- → en uygunu: **liste**
- → kuyruğun **sonu** = listenin ilk elemanı
- → liste işlevleri: insert() ve pop()

Liste işlevleri

Method Name	Use	Explanation
append	alist.append(item)	Adds a new item to the end of a list
insert	alist.insert(i,item)	Inserts an item at the ith position in a list
pop	alist.pop()	Removes and returns the last item in a list
pop	alist.pop(i)	Removes and returns the ith item in a list
sort	alist.sort()	Modifies a list to be sorted
reverse	alist.reverse()	Modifies a list to be in reverse order
del	del alist[i]	Deletes the item in the ith position
index	alist.index(item)	Returns the index of the first occurrence of item
count	alist.count(item)	Returns the number of occurrences of item
remove	alist.remove(item)	Removes the first occurrence of item

Table 1.2: Methods Provided by Lists in Python

Gerçekleme

```
1   class Queue:
2       def __init__(self):
3           self.items = []
4
5       def isEmpty(self):
6           return self.items == []
7
8       def enqueue(self, item):
9           self.items.insert(0,item)
10
11      def dequeue(self):
12          return self.items.pop()
13
14      def size(self):
15          return len(self.items)
```

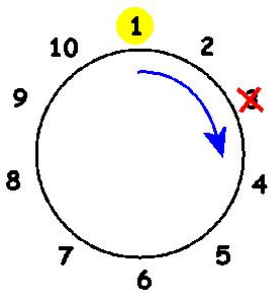

Test - demo

```
>>> execfile("listing_2_9.py")
>>> q = Queue()
>>> q.isEmpty()
True
>>> q.enqueue('dog')
>>> q.enqueue(4)
>>>
>>> q = Queue()
>>> q.isEmpty()
True
>>> q.enqueue(4)
>>> q.enqueue('dog')
>>> q.enqueue(True)
>>> q.size()
3
>>> q.isEmpty()
False
>>> q.enqueue(8.4)
>>> q.dequeue()
4
>>> q.dequeue()
'dog'
>>> q.size()
2
>>>
```


Sıra sizde

- → kuyruğun **başı** = listenin ilk elemanı
- → gerçekleyin

2.4.4 Simulasyon: Hot Potato



- → halka oluştur
- → sırayla say
- → kritik sayıya (ör. "5") denk gelen yanar
- → en son çocuk kalıncaya kadar devam

Hot Potato Oyunu

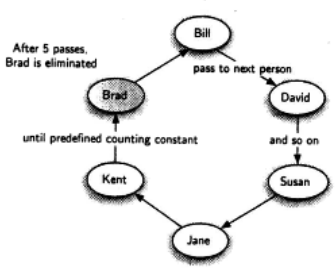
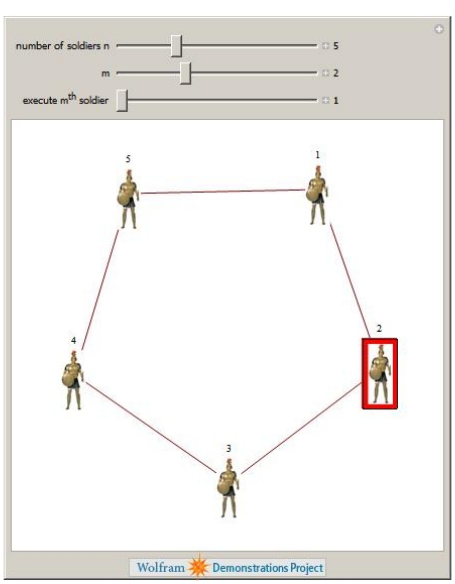


Figure 2.13: A Six Person Game of Hot Potato

- → Josephus problemi olarakta bilinir: http://mathworld.wolfram.com/JosephusProblem.html

Josephus



Kuyruk gerçekleme

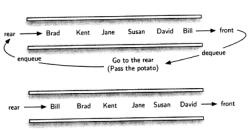


Figure 2.14: A Queue Implementation of Hot Potato

- → kuyruğun başında "Bill", patatesi tutuyor
- → kuyruktan çıkar ve sona gider
- → önündekilerin hepsinin patatesi tutmasını bekleyecektir
- → magic sayıyaya ulaşan yanar ("hot")
- → en sona kalan kazanır

Gerçekleme

```
from listing_2_9 import Queue

def hotPotato(namelist, N):

    simqueue = Queue()
    for name in namelist:
        simqueue.enqueue(name)

    while simqueue.size() > 1:
        for i in range(N):
            simqueue.enqueue(simqueue.dequeue())

        simqueue.dequeue()

    return simqueue.dequeue()

# Test
cocuklar = ['Ali', 'Veli', 'Ayse', 'Suleyman', 'Zehra', 'Ismail']
kazanan = hotPotato(cocuklar, 5)
print "Kazanan cocuk = ", kazanan
```


Test - demo

```
>>> execfile("listing_2_10.py")
>>> hotPotato(["Bill", "David", "Susan", "Jane",
... "Kent", "Brad"], 7)
'Susan'
>>>
```


2.4.5 Simulasyon: yazdırma görevi

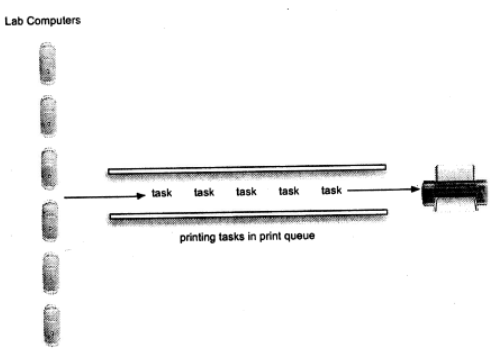


Figure 2.15: Computer Science Laboratory Printing Queue

Neden?

→ yazıcının belli miktar işin üstesinden gelip-gelemeyeceğini test etmek

Gözlem

- → 10 öğrenci/saat
- → 2 yazdırma görevi/saat/öğrenci
- → max: 20 sayfa/görev
- → yazıcı: max: 10 sayfa/dakika (draft)
- → yazıcı: max: 5 sayfa/dakika (normal)
- → Hangi yazdırma kalitesi (sayfa/dk) kullanılmalıdır?

Yapılacaklar

- → Laboratuarı modelleyeceğiz
- → görev (Task) ve yazıcıyı (Task) temsil etmeliyiz
- → öğrencilerin yazdırma istekleri görev kuyruğuna alınacak
- → ortalama olarak yazdırma görevi için ne kadar beklenir?
- → yani ortalama olarak görevin kuyrukta bekleme süresi

Hesap

- → öğrenciler 1 - 20 arasında rastgele adette
- → rastgele zamanda çıktı isteğinde bulunabilir
- → ort. 10 öğrenci
- → ort. 2 kez çıktı isteği
- → ort. 20 görev/saat
- → 20 görev / 1 saat x 1 saat / 3600 sn = 1 görev / 180 sn
- → her bir 180 sn'lik dilimde rastgele bir anda görev ortaya çıkar

2.4.5.1 Ana simülasyon adımları

1. Görev kuyruğunu oluştur. Görev zaman damgası (timestamp). Kuyruk başlangıçta boş.
2. her bir saniye (currentSecond)
   - → yeni görev var mı? Öyleyse currentSecond zaman damgasıyla görevi kuyruğa koy
   - → yazıcı meşgul değilse ve görev bekliyorsa
     - → *yazıcı kuyruğundan görevi (*newtask*) çıkar ve yazıcıya gönder*
     - → *yazıcıyı meşgule al:* currentTask = newtask
     - → *zaman damgasını* currentSecond*'dan çıkararak o görev için* ***bekleme süresini*** *hesapla [...]*

devam

2. her bir saniye (currentSecond)
   - → [...] yazıcı meşgul değilse ve görev bekliyorsa
     - → *[...] daha sonraki hesaplar için bekleme süresini listeye ekle*
     - → *görevdeki sayfa sayısından yazdırma süresini hesapla*
     - → *şu anki görev bir saniyeliğine yazdırılsın,* timeRemaining--
     - → *görev tamamlanınca yazıcı artık meşgul değil:* currentSecond = None
3. Simülasyon tamamlanınca bekleme sürelerinden (waitingtimes) ortalama bekleme süresini hesapla

3.4.5.2 Python Gerçekleme: Printer

```
class Printer:
    def __init__(self, pages):
        self.pagerate = pages
        self.currentTask = None
        self.timeRemaining = 0

    def tick(self):
        if self.currentTask != None:
            self.timeRemaining = self.timeRemaining - 1
            if self.timeRemaining == 0:
                self.currentTask = None

    def busy(self):
        if self.currentTask != None:
            return True
        else:
            return False

    def startNext(self,newtask):
        self.currentTask = newtask
        self.timeRemaining = newtask.getPages() \
                             * 60/self.pagerate
```


Task

```
import random
class Task:
    def __init__(self,time):
        self.timestamp = time
        self.pages = random.randrange(1,21)

    def getStamp(self):
        return self.timestamp

    def getPages(self):
        return self.pages

    def waitTime(self, currenttime):
        return currenttime - self.timestamp
```


Ana simülasyon

```
from queue , printer , task import *
import random

def simulation(numSeconds, pagesPerMinute):

    labprinter = Printer(pagesPerMinute)
    printQueue = Queue()
    waitingtimes = []

    for currentSecond in range(numSeconds):

      if newPrintTask():
         task = Task(currentSecond)
         printQueue.enqueue(task)

      if (not labprinter.busy()) and \
                (not printQueue.isEmpty()):
        nexttask = printQueue.dequeue()
        waitingtimes.append( \
            nexttask.waitTime(currentSecond))
        labprinter.startNext(nexttask)

      labprinter.tick()
```

Ana simülasyon

```
        averageWait=sum(waitingtimes)/float(len(waitingtimes))
        print "Average Wait Time%6.2f seconds"%(averageWait),
        print "Tasks Remaining %3d"%(printQueue.size())


    def newPrintTask():
        num = random.randrange(1,181)
        if num == 180:
            return True
        else:
            return False
```


2.4.5.3 Tartışma

- ⟶ ortalama öğrenci sayısı 20 olursa ne olur?
- ⟶ cumartesi ders yok. beklemeye değer mi?
- ⟶ ortalama görev uzunluğu azalırsa ne olur?

2.5 Deque

- ⟶ Deque doğrusal veri türü

2.5.1 Deque nedir?

- ⟶ çift uçlu, sıralı öğeler
- ⟶ her iki uca ekle/çıkart
- ⟶ hem yığıt hem de kuyruk yeteneği

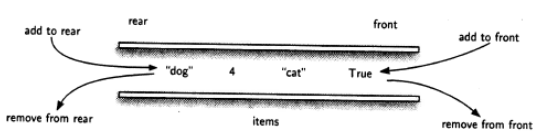


Figure 2.16: A Deque of Python Data Objects

2.5.2 Deque Soyut Veri Türü

Deque(): *boş deque oluştur, parametre gerektirmez*

addFront(item): *deque'in önüne yeni bir öğe ekle. Öğe ister, bir şey döndürmez.*

addRear(item): *deque'in arkasına yeni bir öğe ekle. Öğe ister, bir şey döndürmez.*

removeFront(): *deque'in önünden öğeyi çıkarır. Deque güncellenir.*

removeRear(): *deque'in arkasından öğeyi çıkarır. Deque güncellenir.*

isEmpty(): *deque boş mu?*

size(): *deque'de kaç öğe var?*

Deque işlemleri

Deque Operation	Deque Contents	Return Value
d.isEmpty()	[]	True
d.addRear(4)	[4]	
d.addRear('dog')	['dog',4,]	
d.addFront('cat')	['dog',4,'cat']	
d.addFront(True)	['dog',4,'cat',True]	
d.size()	['dog',4,'cat',True]	4
d.isEmpty()	['dog',4,'cat',True]	False
d.addRear(8.4)	[8.4,'dog',4,'cat',True]	
d.removeRear()	['dog',4,'cat',True]	8.4
d.removeFront()	['dog',4,'cat']	True
d.size()	['dog',4,'cat']	3

Table 2.6: Examples of Deque Operations

2.5.3 Python da gerçekleme

```
1  class Deque:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def addFront(self, item):
9          self.items.append(item)
10
11     def addRear(self, item):
12         self.items.insert(0,item)
13
14     def removeFront(self):
15         return self.items.pop()
16
17     def removeRear(self):
18         return self.items.pop(0)
19
20     def size(self):
21         return len(self.items)
```


Kod açıklama

- → deque'in arkası listenin 0 pozisyonu alınmıştır
- → pop( ) listenin son elemanı: removeFront
- → pop(0) listenin ilk elemanı: removeRear
- → insert : addRear
- → append : addFront

2.5.4 Palindrome denetçisi

→ radar, kelek, madam, toot, ...

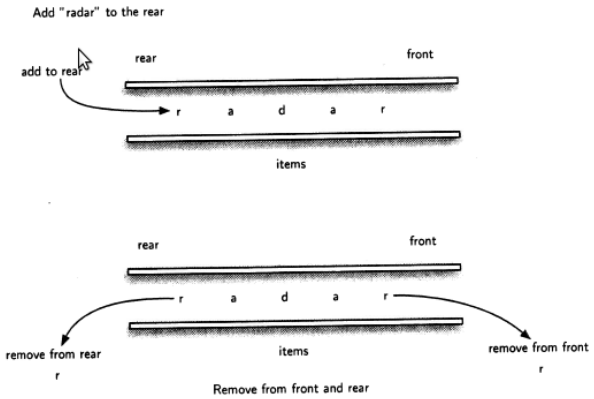


Figure 2.17: A Deque

Kod

```
1  def palchecker(aString):
2
3      chardeque = Deque()
4
5      for ch in aString:
6          chardeque.addRear(ch)
7
8      stillEqual = True
9
10     while chardeque.size() > 1 and stillEqual:
11         first = chardeque.removeFront()
12         last = chardeque.removeRear()
13         if first != last:
14             stillEqual = False
15
16     return stillEqual
```


Kod açıklama

→ soldan sağa tara

→ her bir karakteri deque'in arkasına ekle

→ her iki uçtan öğeleri çek, karşılaştır

→ bir karakter kalırsa palindrome'dur

Özet

→ Doğrusal Veri Yapıları, verileri sıralı tutar

→ Yığıt, basit veri yapılarıdır; LIFO

→ Yığıt: push, pop, isEmpty

→ Kuyruk, basit veri yapılarıdır; FIFO

→ Kuyruk: enqueue, dequeue, isEmpty

→ İfadeler: prefix, infix ve postfix

→ Yığıtlar, ifade dönüşümünde ve hesabında faydalıdır

→ Yığıt, terssel karakteristik

→ Kuyruk, zamanlama simulasyonu

→ Simulasyon, gerçek yaşam durumlarını oluşturmak. "Şöyle olursa ne olur?"

→ Deque hem yığıt hem de kuyruk davranışı

→ Deque: addFront, addRear, removeFront, removeRear, isEmpty

Anahtar Kelimeler

- → Dengeli parantezler
- → Infix, Prefix, Postfix
- → Palindrome
- → Yığıt, Kuyruk, Deque
- → LIFO, FIFO
- → Doğrusal veri yapısı
- → Öncelik
- → Simulasyon