دوره جامع پایتون:

بخش تسلط بر کدنویسی به زبان پایتون

جلسه یازدهم

دکتر ذبیح اله ذبیحی

وراثت

- وراثت توانایی کلاس جدید با استفاده از کلاس های موجود است.
- یکی از مزیت های این ویژگی اضافه کردن متد های جدید به کلاس بدون تعییر در کلاس موجود است.
- کلاس جدید تمام متدهای کلاس موجود را به ارث می برد.
- به کلاس موجود والد به کلاس جدید فرزند یا زیر کلاس گویند.

```
class Person:
  def __init__(self, first_name, last_name):
    self.fname = first_name
    self.lname = last_name

  def printname(self):
    print(self.fname, self.lname)

class Student(Person):
  def __init__(self, first_name, last_name, graduation_year):
    super().__init__(first_name, last_name)
    self.graduationyear = graduation_year

  def welcome(self):
    print("Welcome", self.fname, self.lname, "to the class of", self.graduationyear)
first_name=input("first_name=")
last_name=input("last_name=")
graduation_year=input("graduation_year=")
x = Student(first_name, last_name, graduation_year)
x.welcome()
```

- استفاده از تابع super () در زبان پایتون باعث می شود کلاس فرزند تمامی خصوصیات و متدهای کلاس والد را به ارث ببرد.
- اگر متدی در کلاس فرزند تعریف کنید که نام یکسانی با متدی در کلاس والد داشته باشد، متد کلاس فرزند آن را override می کند (متد کلاس والد برای کلاس فرزند باطل می شود)

لیست پیوندی

- فهرست پیوندی یا لیست پیوندی : ساختاری شامل دنباله‌ای از عناصر است که هر عنصر دارای اشاره‌گری به عنصر بعدی در دنباله است. فهرست پیوندی از جملهٔ ساده‌ترین و رایج‌ترین داده‌ساختارها است و در پیاده‌سازی از داده‌ساختارها پشته، صف و جدول درهم‌سازی استفاده می‌شود. مزیت مهم فهرست پیوندی نسبت به آرایه‌ها این است که ترتیب قرار گرفتن داده‌ها در آن با ترتیب قرار گرفتن آن‌ها در حافظه متفاوت است. به همین دلیل فهرست پیوندی دارای این ویژگی است که درج و حذف گره‌ها در هر نقطه‌ای از فهرست، با تعداد ثابتی از عملیات امکان‌پذیر است. از طرف دیگر فهرست پیوندی اجازه دستیابی تصادفی به داده یا هرگونه اندیس‌گذاری را نمی‌دهد. در نتیجه بسیاری از اعمال ابتدایی نظیر به دست آوردن آخرین عنصر فهرست، پیدا کردن عنصر شامل داده مورد نظر، یا مشخص کردن مکان درج یک عنصر جدید ممکن است نیازمند بررسی اکثر عناصر فهرست باشد.

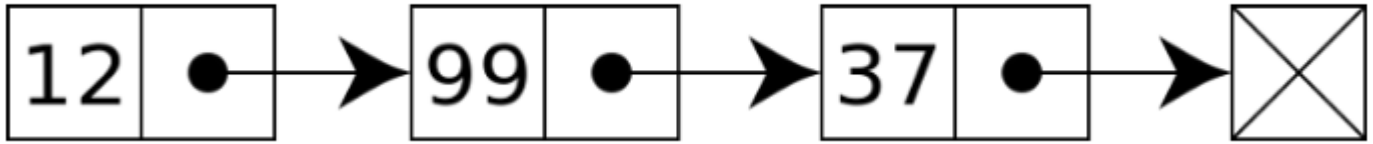
- لیست پیوندی از گره ها تشکیل شده اند که هر گره شامل آدرسی به گره بعدی لیست است.
- هر گره شامل واحده از داده به نام بار است.
- یک لیست پیوندی شامل:

۱- لیست تهی نمایش داده شده با None

۲- یک گره شامل یک شی بار و آدرسی به لیست پیوندی است

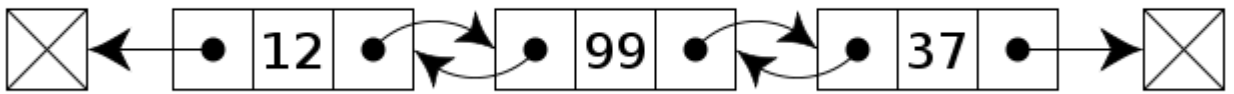
لیست یک طرفه

```
class Node:
    def __init__(self,val):
        self.val = val
        self.next = None
    def n(self):
        node = self # start from the head node
        while node != None:
            print (node.val)
            node = node.next
node1 = Node(12)
node2 = Node(99)
node3 = Node(37)
node1.next = node2 # 12->99
node2.next = node3 # 99->37
node1.n()
```



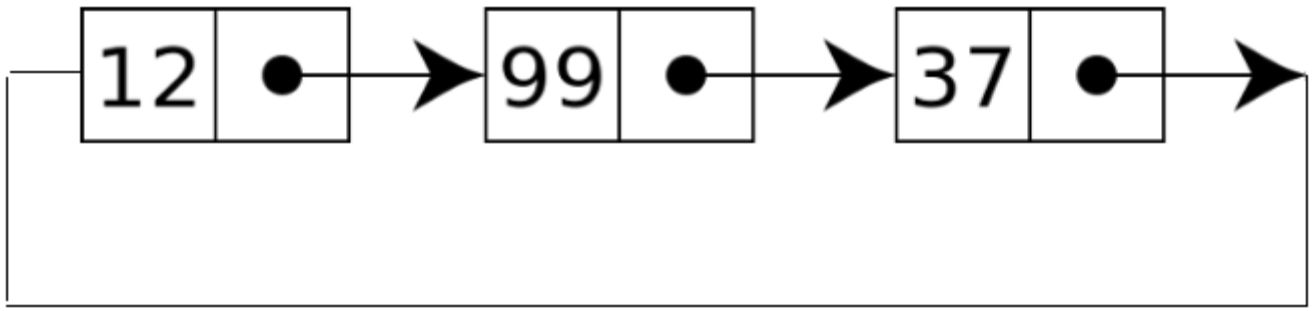
```
class Node:
    def __init__(self,val):
        self.val = val
        self.next = None
    def n(self):
        node = self # start from the head node
        while node != None:
            print (node.val)
            node = node.next
list1=[1,2,3,4,5]
list2=[6,7,8]
list3=[9,10,11,12]
node1 = Node(list1)
node2 = Node(list2)
node3 = Node(list3)
node1.next = node2 # list1->list2
node2.next = node3 # list2->list3
node1.n()
```

لیست دو طرفه



```
class Node:
    def __init__(self,val):
        self.val = val
        self.next = None
        self.prev = None
    def n(self):
        node = self
        while node != None:
            print (node.val)
            node = node.next
    def p(self):
        node = self
        while node != None:
            print (node.val)
            node = node.prev
node1 = Node(12)
node2 = Node(99)
node3 = Node(37)
node1.next = node2 # 12->99
node2.next = node3 # 99->37
node2.prev=node1   # 12<--99
node3.prev=node2   #99<--37
node1.n()
node3.p()
```

لیست حلقه ای (دایره ای)



```
class Node:
    def __init__(self,val):
        self.val = val
        self.next = None
        self.prev = None
    def n(self):
        node = self
        i=0
        while node != None:
            if (i<15):
                print (node.val)
                node = node.next
                i=i+1


node1 = Node(12)
node2 = Node(99)
node3 = Node(37)
node1.next = node2 # 12->99
node2.next = node3 # 99->37
node3.next=node1
node1.n()
```

- ساختمان داده‌ها روش‌های ذخیره داده‌ها در رایانه با هدف دسترسی آسان‌تر و بهینه تر است.
- در حالیکه الگوریتم روشی به منظور حل مسئله به وسیله کامپیوتر است.

پرکاربرد ترین ساختمان داده ها

- آرایه Array:

عدادی متغیر از یک نوع داده و تحت یک نام می‌باشد. هر یک از متغیرهای درون آرایه با یک شماره که به آن «اندیس» می‌گوییم از یکدیگر متمایز می‌شوند. متغیرهای درون آرایه را «عناصر آرایه» می‌نامند که همگی قابلیت نگهداری فقط یک نوع داده را دارند.

- صف Queue:

خروج به ترتیب ورود یکی از روش‌های سازماندهی کنترل داده با توجه به زمان و اولویت‌بندی است.

- پشته Stack:

عناصر پشته فقط از یک طرف (بالای پشته) قابل دستیابی اند.

- لیست پیوندی Linked list

ساختاری شامل دنباله‌ای از عناصر است که هر عنصر دارای اشاره‌گری به عنصر بعدی در دنباله است.

- گراف Graph:

داده‌ساختاری انتزاعی است که به صورت گراف جهت دار و بدون جهت پیاده‌سازی می‌شود و. هدفش به کارگیریِ مفهوم گراف از ریاضیات و به خصوص نظریه گراف است.

- درخت Tree:

شبیه به یک ساختار درختی با مجموعه‌ای از گره‌های متصل به هم است. درخت یک گراف همبند بدون دور است. اکثر نویسندگان این قید را نیز اضافه می‌کنند که گراف باید بدون جهت باشد. به علاوه بعضی قید بدون وزن بودن یال‌ها را نیز اضافه می‌کنند.

- جدول درهم‌سازی Hash table

پشته

- پشته ساختمان داده ای است که از لیست برای سازماندهی داده ها استفاده میکند و در عین حال از انتزاع نیز پشتیبانی میکند و یک نوع داده انتزاعی را فراهم میسازد. در پشته عمل اضافه کردن وحذف عنصر، فقط در یک طرف آن، بنام بالای پشته انجام میشود. یعنی عنصری که از همه دیرتر وارد پشته شد، از همه زودتر از پشته حذف میگردد. بهمین دلیل گفته میشود که پشته از سیاست خروج به ترتیب عکس ورود LIFO پیروی میکند.

- __init__

مقدار دهی اولیه

- push

اضافه کردن یک عضو جدید به پشته

- Pop

حذف و برگرداندن یک عضو از پشته. عضوی که برگردانده می شود همیشه آخرین عضو اضافه شده است.

- isEmpty

بررسی اینکه ایا آیا پشته تهی است یا خیر

```
class stack():
    def __init__(self):
        self.items=[]
    def push(self,item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def isEmpty(self):
        return (self.items==[])

s=stack()
s.push("ali")
s.push("neda")
while not s.isEmpty():
    print(s.pop())
```

```
stack = []

stack.append('a')
stack.append('b')
stack.append('c')

print('Initial stack')
print(stack)

print('\nElements poped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are poped:')
print(stack)
```

صف

- خروج به ترتیب ورود یکی از روش‌های سازماندهی کنترل داده با توجه به زمان و اولویت‌بندی است. روشی که به هر فرایندی زمانی از زمان پردازنده را مطابق با ترتیب ورودش اختصاص می‌دهد.
- هر مهره‌ای که زودتر وارد شود، زود تر بررسی می‌گردد و هر مهره‌ای پس از آن وارد شود صبر می‌کند تا اعمال انجام گرفته روی مهره اول تمام شود.
- این موضوع شبیه رفتار صف بندی انسان‌ها است، جاییکه افراد صف را به ترتیب ورودشان ترک می‌نمایند.
- LIFOیا «آخرین ورودی، اولین خروجی»
- FILOیا«اولین ورودی، آخرین خروجی»
- هر دو حالت خاصی از یک لیست عام هستند. تفاوت در داده‌ها وجود ندارد. بلکه در قواعد برای دستیابی به محتوا است.

```
queue = []

queue.append('a')
queue.append('b')
queue.append('c')

print("Initial queue")
print(queue)

print("\nElements dequeued from queue")
print(queue.pop(0))
print(queue.pop(0))
print(queue.pop(0))

print("\nQueue after removing elements")
print(queue)
```

تفاوت پشته و صف

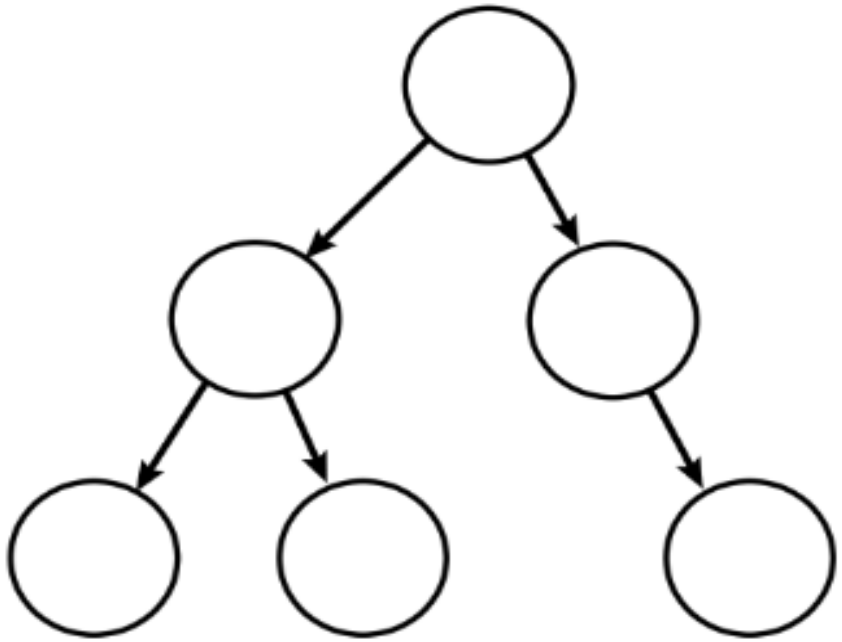
- دستهٔ کاغذها روی میز، مثالی خوب از پشته‌است. در این حالت ما تنها می‌توانیم بر روی دستهٔ کاغذها، کاغذی بگذاریم و از طرفی تنها می‌توانیم از روی دستهٔ کاغذها، کاغذی برداریم (یعنی ورود و خروج از یک سمت انجام می‌گیرد). روشن است که در این حالت آخرین کاغذی که روی دسته کاغذها قرار داده شده، نخستین کاغذی است که برداشته می‌شود و اولین کاغذی که روی میز گذاشته شده، آخر از همه برداشته خواهد شد.
- صف نانوایی، مثالی خوب از صف است. در این حالت، برخلاف پشته، آدم‌ها به ته صف اضافه می‌شوند و از سر صف خارج می‌شوند (یعنی ورود و خروج از دو سمت متمایز انجام می‌گیرد). به این ترتیب روشن است که آخرین کسی که وارد صف شده، آخرین کسی است که نان دریافت می‌کند و اولین کسی که وارد صف شده، نخستین فردی است که نان می‌گیرد.

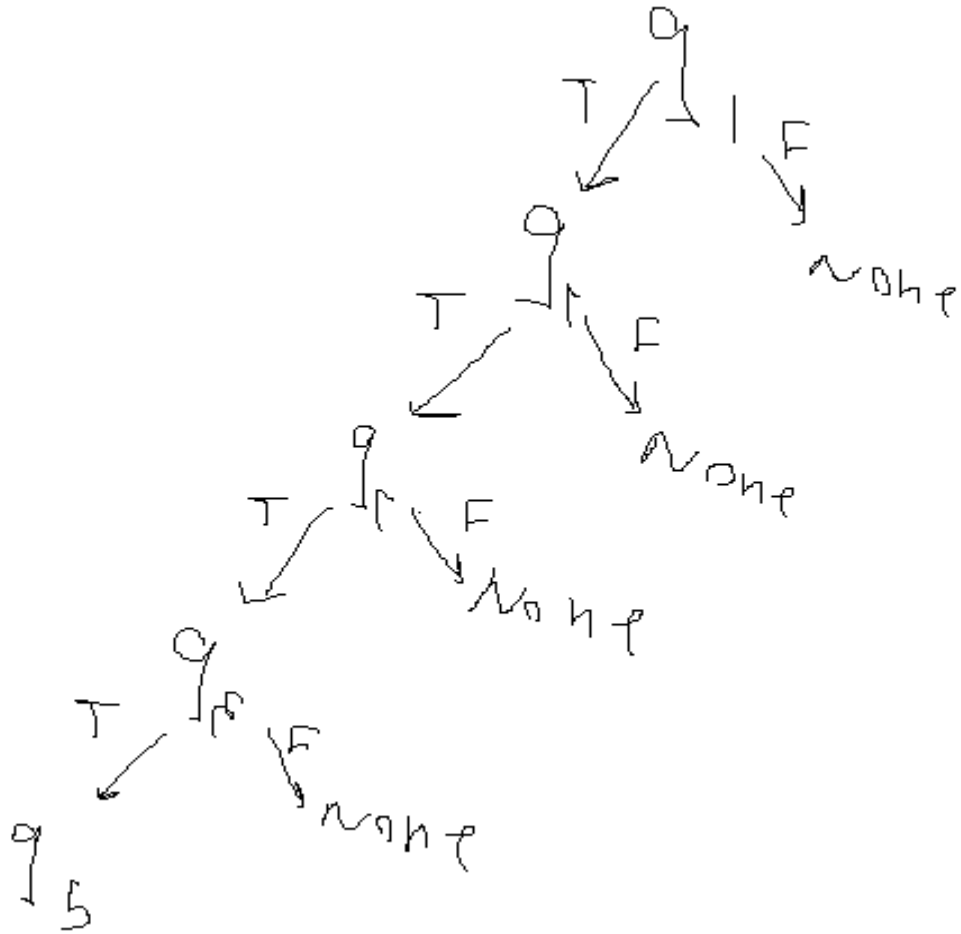
درخت

- بالای درخت ریشه نام دارد
- گره ها شاخه نامیده می شوند.
- گره هایی که در انتها قرار گرفته اند و به None ختم می شوند برگ نامیده می شوند.
- ----------

- گروه فوقانی پدر یا والد و گره هایی که والد به آن اشاره می کند فرزند نامیده می شوند. گره هایی با پدر واحد برادر نامیده می شوند. تمام گره هایی که فاصله یکسانی از ریشه دارند یک سطح را تشکیل می دهند.
- بالا والد/ریشه
- پایین فرزندان/برگها

درخت دودویی



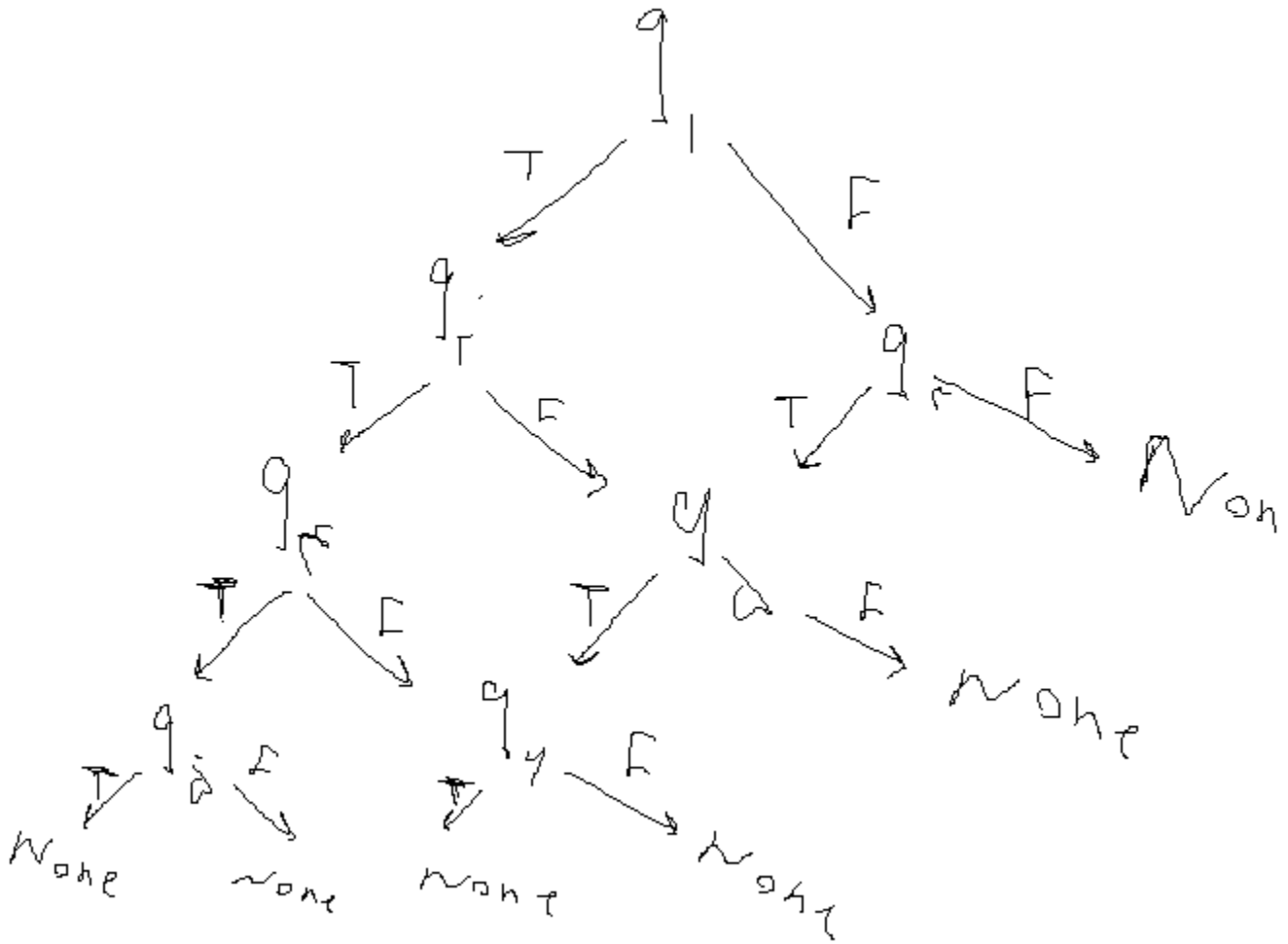


```
class Node:
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right=None
    def n(self):
        node = self
        score=0
        while node != None:
            print (node.val)
            s1=input("answer?")
            s2=input("Ture or False (T or F)=")
            if s2=="T":
              node = node.left
              score=score+1
              print("score=",score)
              print("----------------------")
            else:
              node = node.right
              print("score=",score)
              print("--------Finish--------------")
            if node==None:
               print("--------Finish--------------")
```

```
q1="q1:2+5=?"
q2="q2:2-4=?"
q3="q3:2*3=?"
q4="q4:6/3=?"
q5="q5:(6/2)-2=?"
node1=Node(q1)
node2=Node(q2)
node3=Node(q3)
node4=Node(q4)
node5=Node(q5)
node1.left=node2
node2.left=node3
node3.left=node4
node4.left=node5
node1.n()
```

تمرین

- کد درخت زیر را بنویسید



بازی دوز

```
theBoard = {'7': ' ' , '8': ' ' , '9': ' ' ,
            '4': ' ' , '5': ' ' , '6': ' ' ,
            '1': ' ' , '2': ' ' , '3': ' ' }

board_keys = []

for key in theBoard:
    board_keys.append(key)



def printBoard(board):
    print(board['7'] + '|' + board['8'] + '|' + board['9'])
    print('-+-+-')
    print(board['4'] + '|' + board['5'] + '|' + board['6'])
    print('-+-+-')
    print(board['1'] + '|' + board['2'] + '|' + board['3'])
```

```
def game():

    turn = 'X'
    count = 0



    while count<10:
        printBoard(theBoard)
        print("It's your turn," + turn + ".Move to which place?")

        move = input()

        if theBoard[move] == ' ':
            theBoard[move] = turn
            count += 1
        else:
            print("That place is already filled.\nMove to which place?")
            continue
```

```
if count >= 5:
        if theBoard['7'] == theBoard['8'] == theBoard['9'] != ' ':
            printBoard(theBoard)
            print("\nGame Over.\n")
            print(" **** " +turn + " won. ****")
            break
        elif theBoard['4'] == theBoard['5'] == theBoard['6'] != ' ':
            printBoard(theBoard)
            print("\nGame Over.\n")
            print(" **** " +turn + " won. ****")
            break
        elif theBoard['1'] == theBoard['2'] == theBoard['3'] != ' ':
            printBoard(theBoard)
            print("\nGame Over.\n")
            print(" **** " +turn + " won. ****")
            break
        elif theBoard['1'] == theBoard['4'] == theBoard['7'] != ' ':
            printBoard(theBoard)
            print("\nGame Over.\n")
            print(" **** " +turn + " won. ****")
            break
```

```
elif theBoard['2'] == theBoard['5'] == theBoard['8'] != ' ':
    printBoard(theBoard)
    print("\nGame Over.\n")
    print(" **** " +turn + " won. ****")
    break
 elif theBoard['3'] == theBoard['6'] == theBoard['9'] != ' ':
    printBoard(theBoard)
    print("\nGame Over.\n")
    print(" **** " +turn + " won. ****")
    break
 elif theBoard['7'] == theBoard['5'] == theBoard['3'] != ' ':
    printBoard(theBoard)
    print("\nGame Over.\n")
    print(" **** " +turn + " won. ****")
    break
 elif theBoard['1'] == theBoard['5'] == theBoard['9'] != ' ':
    printBoard(theBoard)
    print("\nGame Over.\n")
    print(" **** " +turn + " won. ****")
    break
```

```
        if count == 9:

            printBoard(theBoard)
            print("\nGame Over.\n")
            print("It's a Tie!!")
            break



        if turn =='X':
            turn = 'O'
        else:
            turn = 'X'

    restart = input("Do want to play Again?(y/n)")
    if restart == "y" or restart == "Y":
        for key in board_keys:
            theBoard[key] = " "

        game()

if __name__ == "__main__":
    game()
```

بازی سنگ کاغذ قیچی

- قیچی مقدار ۳، کاغذ مقدار ۲، سنگ مقدار ۱ دارد
- اگر انتخاب دو بازیکن یکسان باشد، بازی تکرار می شود. اگر متفاوت باشد بازی یک برنده دارد و برنده یک امتیاز دارد
- قیچی در مقابل کاغذ برنده است چون قیچی کاغذ را می برد.
- سنگ در برابر قیچی برنده است است چون سنگ قیچی را میشکند.
- کاغذ، سنگ را می برد چون کاغذ دور سنگ می پیچد.
- بازی را پنج باز تکرار کنید و برنده را اعلام کنید.

```
import random
print("'scissor' : 3, 'paper' : 2, 'stone' : 1"   )
game = {'scissor' : 3, 'paper' : 2, 'stone' : 1}
n=int(input("number of game="))
i=1
score_player1=0
score_player2=0
list1=[1,2,3]
```

```
while i <= n:
  print('number of match = ', i)
  player1 = int(input('player1 choose a match : '))
  player2 =random.choice(list1)
  print('player2 choose a match = ',player2)
  if player1 == player2:
    print('try again')
    continue
  elif player1 == 3 and player2 == 2:
    print('player1 win')
    score_player1=score_player1+1

  elif player1 == 1  and player2 == 3:
    print('player1 win')
    score_player1=score_player1+1
  elif player1 == 2 and player2 == 1:
    print('player1 win')
    score_player1=score_player1+1
  else:
   print('player2 win')
   score_player2=score_player2+1

  print("score_player1",score_player1)
  print("score_player2",score_player2)
  print('-------------------------')
  i = i + 1
```

```
if score_player1>score_player2:
    print("player1 won")
elif score_player2>score_player1:
    print("player2 won")
else:
    print("equal“)
```

مثال تبدیل کد مورس به متن

```
message=.... . .-.. .-.. --- .-.-.-  --. ---  .... --- -- . ""
message=message+ ' '
decipher = ''
citext = ''
for letter in message:
    if (letter != ' '):
        i = 0
        citext=citext+ letter
    else:
        i=i+1
        if i == 2 :
         decipher=decipher+' '
        else:
         decipher=decipher+ list(MORSE_CODE_DICT.keys())[list(MORSE_CODE_DICT.values()).index(citext)]
        citext = ''
print(decipher)
```