Objectives

- To introduce the string data type
- To demonstrate the use of string methods and operators
- To introduce simple cryptographic algorithms

String

- A sequence of characters
- Quote delimited
- ‘ single
- “ double
- ‘ ‘ ‘ triple

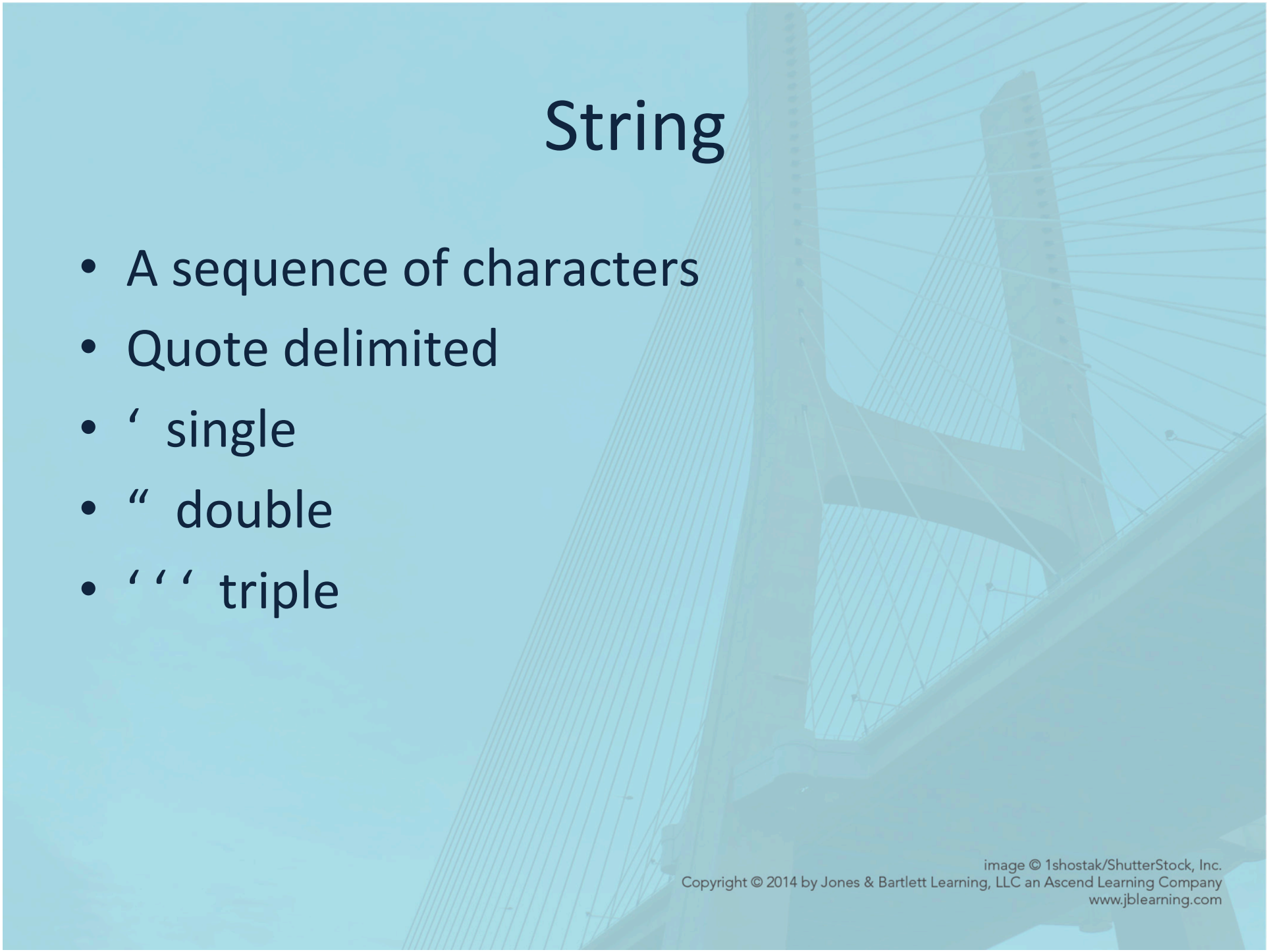


Figure 3.1

Positive indexes	0	1	2	3	4	5	6	7	8	9	10	11
String	P	Y	T	H	O	N		R	O	C	K	S
Negative indexes	–12	–11	–10	–9	–8	–7	–6	–5	–4	–3	–2	–1

Operators

- Concatenation +
- Repetition *
- Indexing [ ]
- Slicing [:]

String Methods

- upper
- lower
- center
- count
- index
- find
- replace

Character Functions

- ord
- char
- Convert between characters and numbers

Listing 3.1

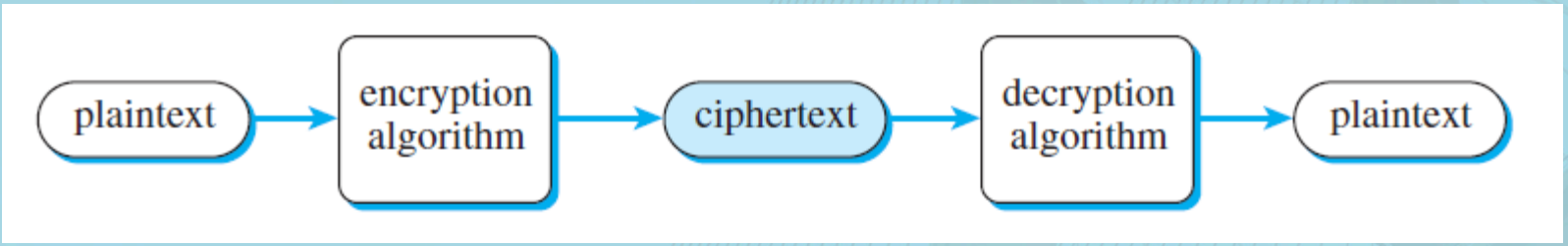
```
def letterToIndex(ch):
    alphabet = "abcdefghijklmnopqrstuvwxyz "
    idx = alphabet.find(ch)
    if idx < 0:
       print ("error: letter not in the alphabet", ch)
    return idx

  def indexToLetter(idx):
    alphabet = "abcdefghijklmnopqrstuvwxyz "
    if idx > 25:
       print ('error: ', idx, ' is too large')
       letter = ''
    elif idx < 0:
       print ('error: ', idx, ' is less  than 0')
       letter = ''
    else:
       letter = alphabet[idx]
    return letter
```

Cryptography

- Encoding and Decoding Messages
- Ciphertext
- Plaintext
- Encryption
- Decryption

Figure 3.2



Transposition Cipher

- Rail Fence
- Even Odd Shuffle

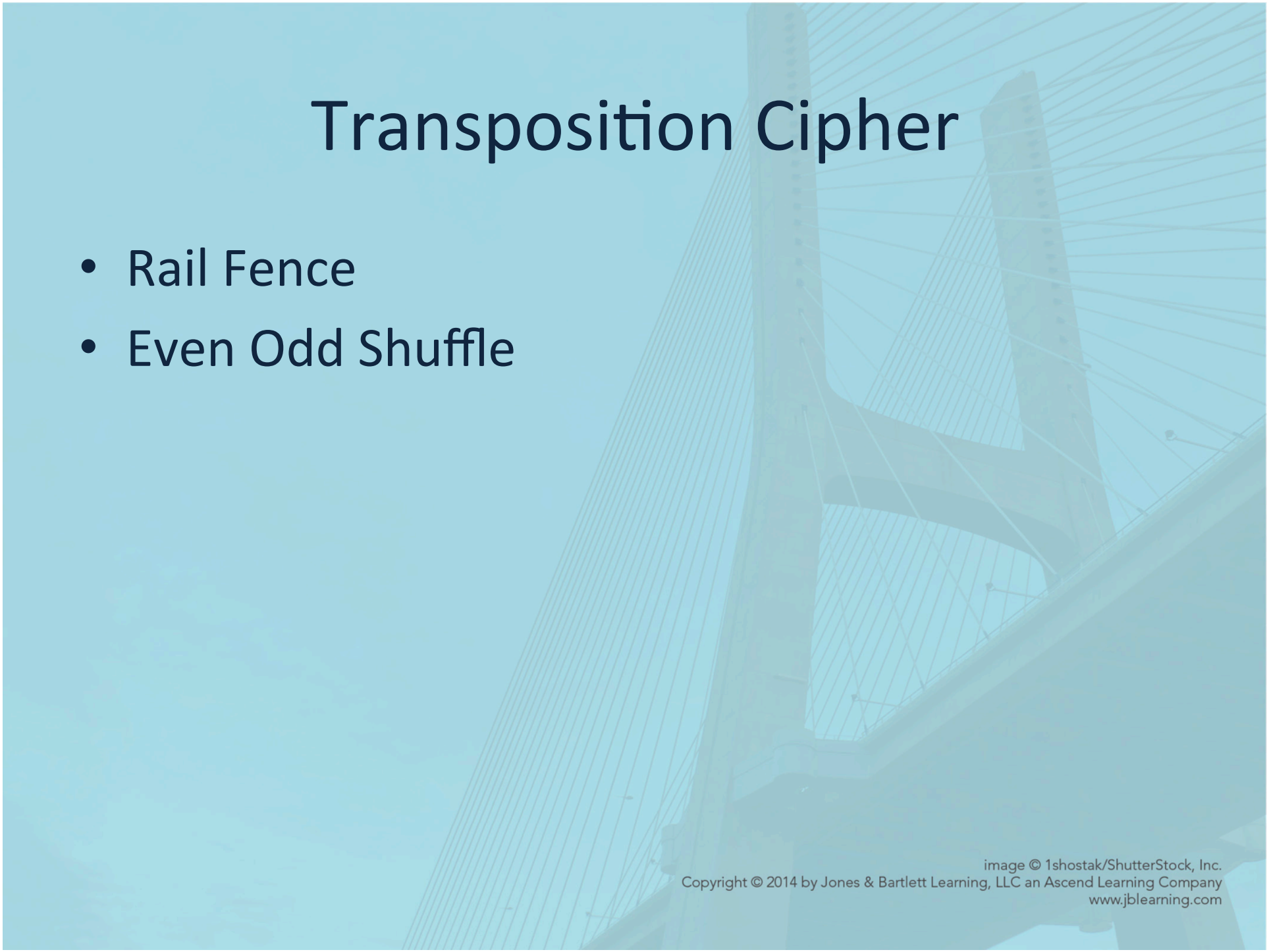
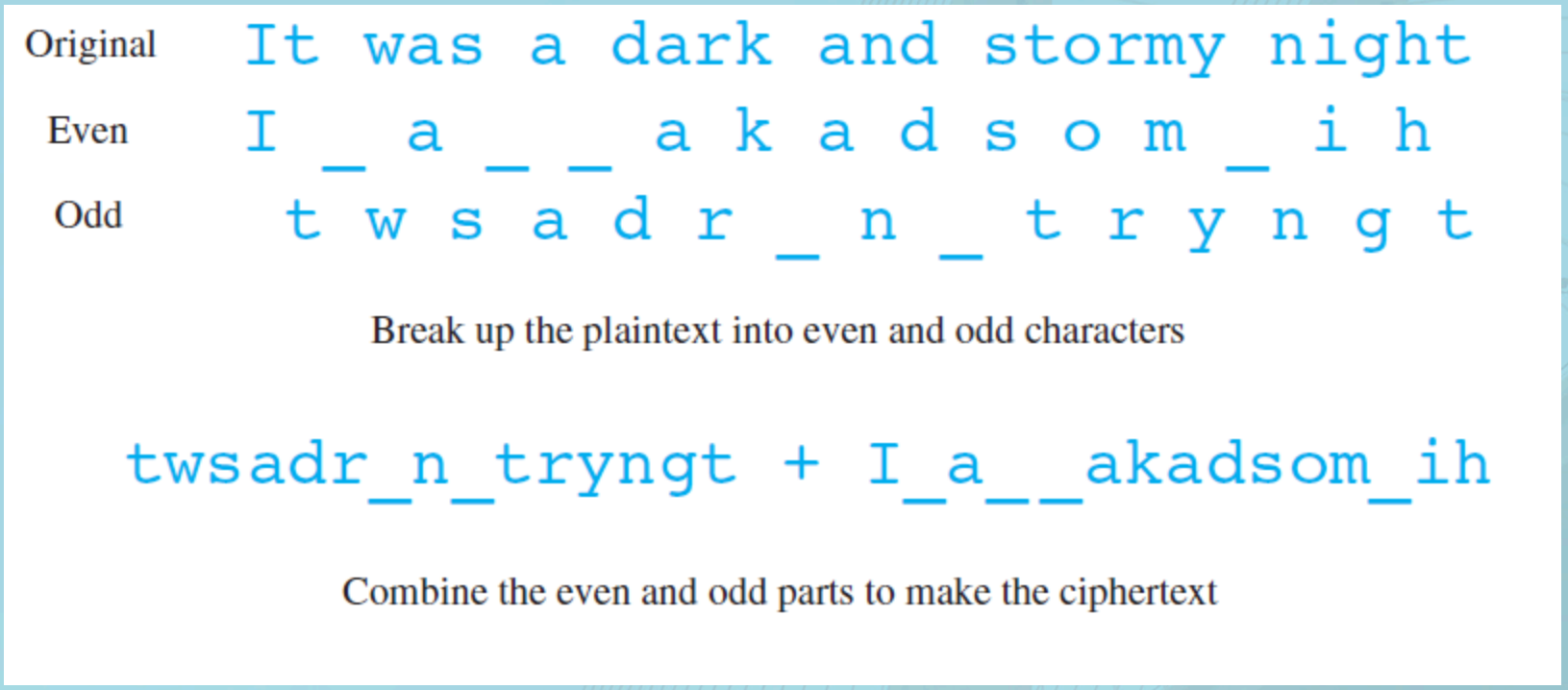


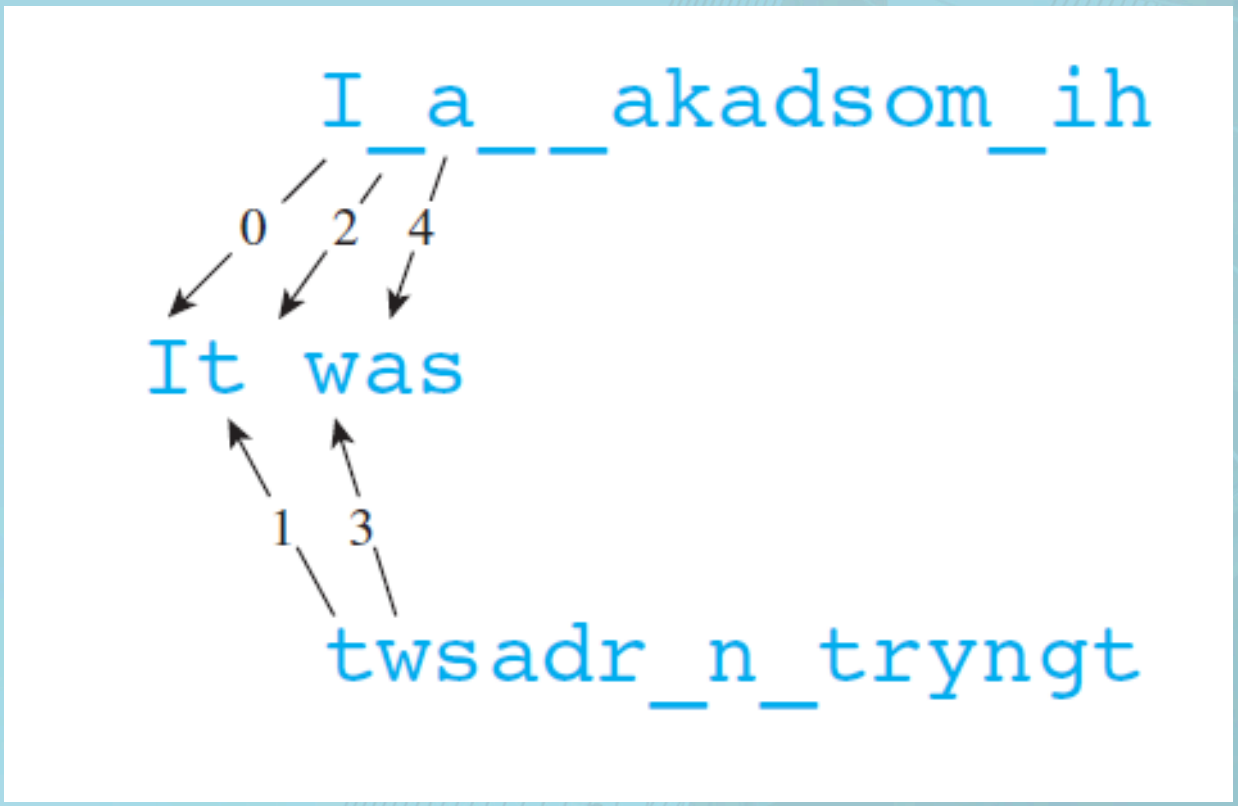
Figure 3.3



Listing 3.2

```
def scramble2Encrypt(plainText):
    evenChars = ""
    oddChars = ""
    charCount = 0
    for ch in plainText:
        if charCount % 2 == 0:
            evenChars = evenChars + ch
        else:
            oddChars = oddChars + ch
        charCount = charCount + 1
    cipherText = oddChars + evenChars
    return cipherText
```

Figure 3.4



Listing 3.3

```
def scramble2Decrypt(cipherText):
    halfLength = len(cipherText) // 2
    oddChars = cipherText[:halfLength]
    evenChars = cipherText[halfLength:]
    plainText = ""

    for i in range(halfLength):
        plainText = plainText + evenChars[i]
        plainText = plainText + oddChars[i]

    if len(oddChars) < len(evenChars):
        plainText = plainText + evenChars[-1]

    return plainText
```

Listing 3.4

```
def encryptMessage():
    msg = input('Enter a message to encrypt: ')
    cipherText = scramble2Encrypt(msg)
    print('The encrypted message is: ', cipherText)
```

Substitution Cipher

- Requires a key
- Substitute one letter for another throughout the entire message

Figure 3.5

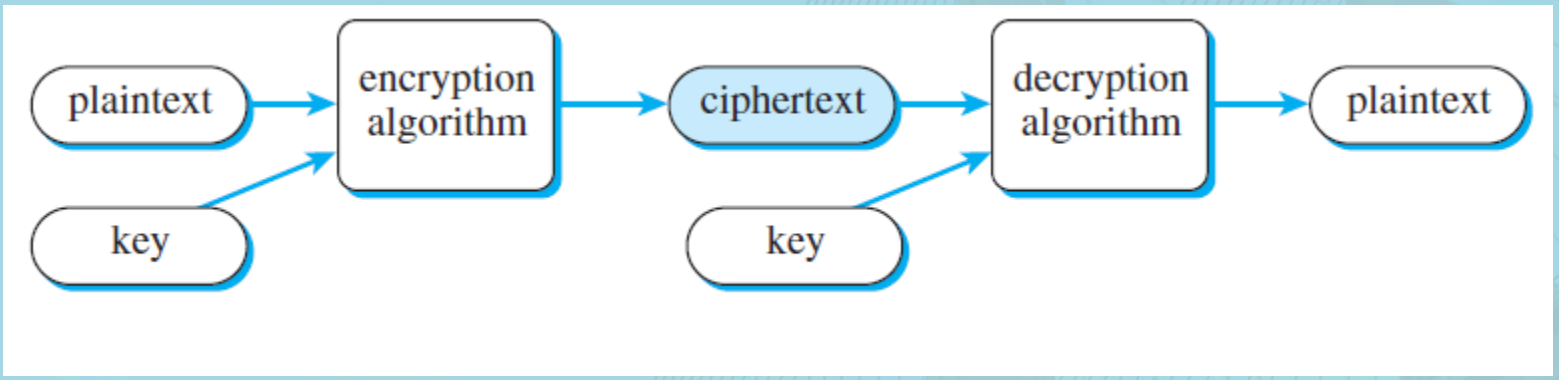
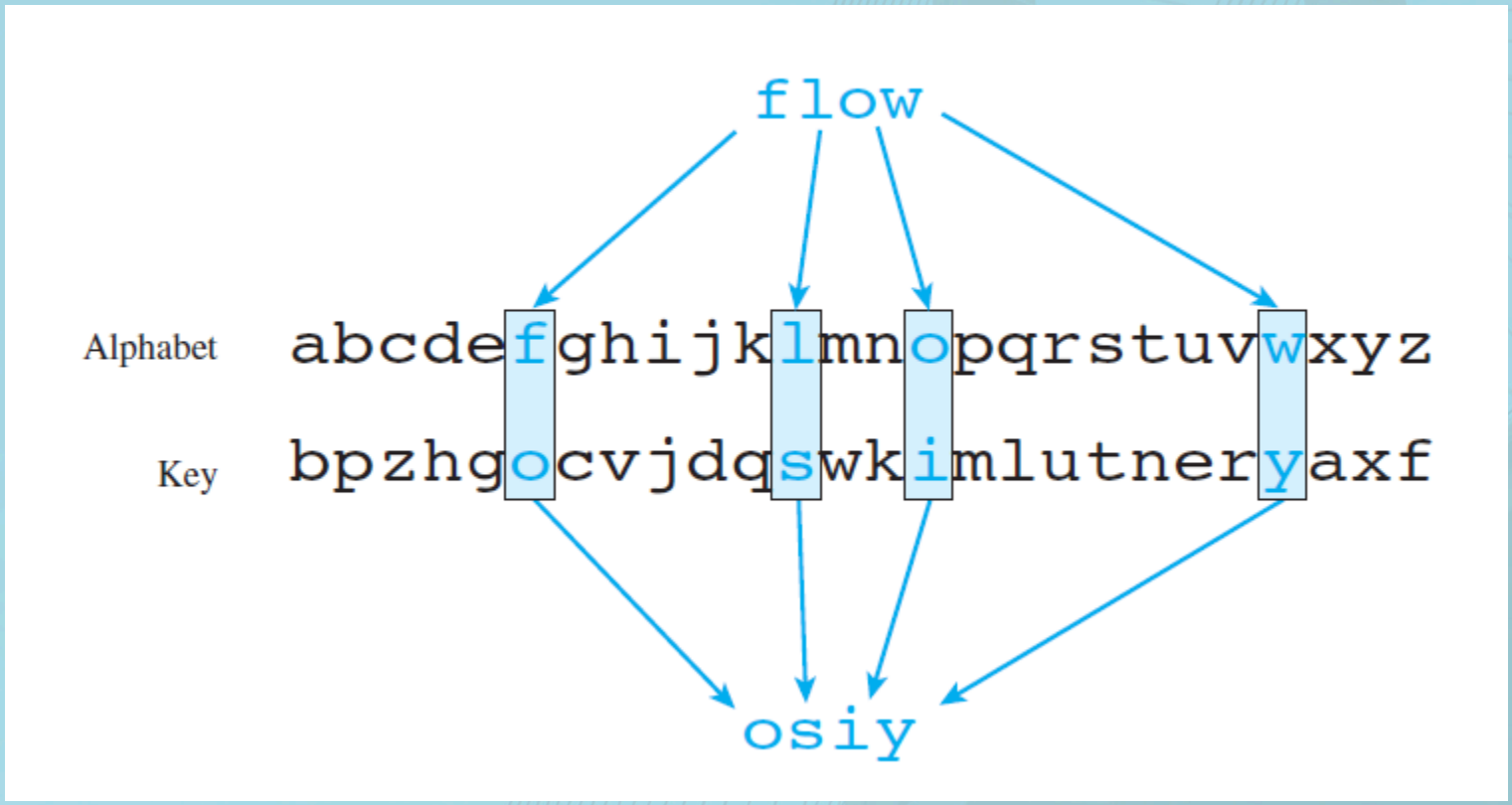


Figure 3.6



Listing 3.5

```
def substitutionEncrypt(plainText,key):
    alphabet = "abcdefghijklmnopqrstuvwxyz "
    plainText = plainText.lower()
    cipherText = ""
    for ch in plainText:
        idx = alphabet.find(ch)
        cipherText = cipherText + key[idx]
    return cipherText
```

Creating a Key

- Random reordering of the alphabet
- Base key on short word or phrase

Listing 3.6

```
def removeChar(string,idx):
    return string[:idx] + string[idx+1:]
```

Listing 3.7

```
def keyGen():
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    key = ""
    for i in range(len(alphabet)):
        ch = random.randint(0,25-i)
        key = key + alphabet[ch]
        alphabet = removeChar(alphabet,ch)
    return key
```

Listing 3.8

```
def removeDupes(myString):
    newStr = ""
    for ch in myString:
        if ch not in newStr:
            newStr = newStr + ch
    return newStr
```

Listing 3.9

```
def removeMatches(myString,removeString):
    newStr = ""
    for ch in myString:
        if ch not in removeString:
            newStr = newStr + ch
    return newStr
```

Listing 3.10

```
def genKeyFromPass(password):
    key = 'abcdefghijklmnopqrstuvwxyz'
    password = removeDupes(password)
    lastChar = password[-1]
    lastIdx = key.find(lastChar)
    afterString = removeMatches(key[lastIdx+1:],password)
    beforeString = removeMatches(key[:lastIdx],password)
    key = password + afterString + beforeString
    return key
```

Vignere Cipher

- Use a different key for each letter
- Vignere Square
- Use a special key for shift

Listing 3.11

```
def vignereIndex(keyLetter,plainTextLetter):
    keyIndex = letterToIndex(keyLetter)
    ptIndex = letterToIndex(plainTextLetter)
    newIdx = (ptIndex + keyIndex) % 26
    return indexToLetter(newIdx)
```

Listing 3.12

```
def encryptVignere(key,plainText):
    cipherText = ""
    keyLen = len(key)
    charNum = 0
    for i in range(len(plainText)):
        ch = plainText[i]
        if ch == ' ':
            cipherText = cipherText + ch
        else:
            cipherText = cipherText + vignereIndex(key[i%keyLen],ch)
    return cipherText
```