```
    else:
        pr = ????
print("探索に失敗しました。")
```

入力：5 7 15 28 29 31 39 58 68 70 95
　　39
出力：探索成功：配列の 7 番目です。

【考え方】
　まずは、二分探索の図と説明をよく見直して、入力例に対してどのように pl，pc，pr などのパラメータが変化するのかをよく考えてみましょう。頭の中で考えるだけでなく、紙に書きながら整理して考えると良い。いきなり pl とか pc のような変数で考えるのが難しい場合には、まずは pl や pc に具体的な数を入れて考えて、それからどの変数が当てはまるのかを考えると良い。そして、????に入る式を考えて、プログラムコードに記入して実行し、うまく動くかどうかを確認して、うまく動かなければ、なぜうまく動かないのか、実際に自分のプログラムに入力する値を入れて手で計算しながら動かしてみると良い。この程度の長さのプログラムであれば、そのように「紙と鉛筆を使って実際にプログラムの動きをなぞってみる」という検証は有効であり、そのような経験を通してプログラミングの考え方を身につけることになる。
　「インデックス」と「リストの要素」を混同する答案が多い（インデックスを書くべきところに要素を書く、あるいはその逆）ので、注意すること。たとえば、list[i]は、list という名前のリストの「インデックス」が i の位置に入っている「リストの要素」を意味することになる。たとえば、

list = [5 7 15 28 29 31 39 58 68 70 95]

のときに、

list[0] = 5
list[1] = 7
list[2] = 15

となる。わからない人は、「コンテナのデータ型」の授業をよく復習すること。

【発展：ハッシュ探索と辞書】
　二分探索よりもさらに効率の良い探索方法に「ハッシュ探索」がある。探索時間を比較すると、線形探索はデータの長さに比例し（O(n)）、二分探索はデータの長さの対数に比例し（O(logn)）、ハッシュ探索はデータの長さに関わらず一定である（O(1)）。ハッシュ探索では、データからハッシュ関数（任意のデータから、別の値を得る関数）によって得られる「ハッシュ」（ハッシュ値）を計算して、そのハッシュからハッシュ表（ハッシュテーブル）によって定まるメモリの場所にデータを格納する、というアルゴリズムである。最初から決められた場所にデータを格納するため、探索する時間はその場所を計算する時間だけとなり、データの長さに依存しない。
　検索エンジンで検索をすると、一瞬で検索した文字列が含まれるウェブサイトの一覧が得られるのは、検索エンジンのクローラがウェブサイトを取得してキャッシュとして保存し、インデックスを作るからである。このときにハッシュ表を使うことで、膨大なキャッシュから目標とする検索文字列が含まれるインデックスを速やかに探し当てることができる。

　Python では、辞書型でハッシュ表によってデータが管理されている。コンテナのデータ型については、リストを中心に扱ってきたが、辞書型について簡単に説明する。辞書型のデータは、たとえば

dict = {'key1': 12, 'key2': 24, 'key3' : 135}

のように「キー」と値によってデータを管理する。そして、dict[‘key1’]のようにキーをインデックスとして値を得る。リストやタプルのようなシーケンス型では、要素の「何番目か」という数をインデックスとして値を得る。それに対して、このようにキー（数とは限らない）をインデックスとして値を得るデータ構造を、一般に「連想配列(associative array)」「辞書」「マップ」などと呼ぶ。すなわち「キーと値の組」のデータ構造である。

Python では、辞書型でこの連想配列が実現されていて、ハッシュ探索を使ってキーから速やかに値を得ることができる。その仕組みは、次のようになっている。キーはハッシュ関数によってハッシュ表の大きさにあわせた整数のハッシュに変換され（ここで使われるハッシュ関数は、暗号で使われるような複雑なものではない）、そのハッシュが定めるメモリの場所にデータが格納される。ハッシュ表の大きさは 2 の n 乗となっている。たとえば、ハッシュ表の大きさが 128 の場合は、６０から 127 までの整数のハッシュが得られ、キーから計算されるハッシュが 58 であれば、58 番目のデータに直接アクセスする。ここで、異なるデータで同じハッシュが計算される「ハッシュの衝突」が生じた場合には、オープンアドレス法というアルゴリズムによって「次に開いている場所」が探索されて、そこに格納される。ハッシュ表のあいている場所が少なくなると、ハッシュ表が 2 倍に拡大され、再構築(rehash)される。詳しいハッシュの計算方法については、Python ソースコードの辞書オブジェクト（dictobject.c）[1]にコメントで解説されている。

このように、Python の辞書型はハッシュ表によってデータが管理されているため、大量のデータを取り扱う場合でもキーから効率的にデータを探索、読み出し、書き込み、追加、削除することができる。また、Python の辞書型でキーとして使えるのは「ハッシュ可能なオブジェクト」に限られる。数値や文字列はハッシュ可能であるが、リストはハッシュ可能でないのでキーとして使うことができない。変更可能なオブジェクト（mutable object）はハッシュ不可能である。タプルは変更不可能でハッシュ可能なので、リストを「キー」として使いたい場合には、タプルに変換するのが良いであろう。

[1] https://github.com/python/cpython/blob/master/Objects/dictobject.c

第 12 回　オブジェクト指向プログラミング

オブジェクト指向プログラミングの考え方について学ぶ。

【オブジェクト指向プログラミング】
Python はオブジェクト指向プログラミングができる。オブジェクト指向プログラミングとは、データとメソッドをひとつにまとめてオブジェクトとして、オブジェクトを中心にプログラミングをすることである。オブジェクト指向プログラミングに特有の用語について、簡単に解説する。
オブジェクトの種類を「クラス」と呼び、あるクラスに基づいて生成された具体的なオブジェクト（物）を「インスタンス」と呼ぶ。たとえば、「ビールとはこういうものである」ということが書かれている酒税法は「ビールクラス」を定めていて、実際に店で売られている１つ１つの缶ビールは、「ビールクラス」に基づいて生成された「ビールインスタンス」（ビールオブジェクト）である、と考えることができる。
Python では、組み込みのクラスとしてたとえば「int クラス」「str クラス」のようなものが定められている。さらに、そのクラスが持つ状態（変数）や、そのクラスがどういう動作をするか（メソッド）が定義されている。そして、あるクラスに基づいて生成されたオブジェクト（インスタンス）は、呼び出されたクラスで定義されているメソッドを使って、操作することができる。
たとえば a = “Hello” という文を実行することで、a という変数に「Hello という文字列が代入される」とこれまで説明をしてきたが、これは str クラスを元に Hello という具体的な文字列の「オブジェクト（インスタンス）」が生成された、ということになる。この Hello という文字列は、str クラスを元に生成されたオブジェクトであるため、str クラスで定義されているメソッドを使うことができる。たとえば、str クラスには replace というメソッドが定義されているので、Hello という文字列に replace というメソッドを適用することができる。なお、replace というメソッドは文字列の置換をするメソッドであり、たとえば次のプログラムの出力は

```
s = "Hello"
print(s.replace("H", "h"))
```

出力：hello

となる。

これまでに使ったメソッドには、str クラスに定義されている split というメソッドがある。これは、文字列をスペースやタブなどの「ホワイトスペース」で区切ったリストを生成するものであり、オプションによって区切り文字を指定することもできるが、詳しくは触れない。これまでのプログラムで使われていた

```
list = list(map(int,input().split()))
```

という文は、input().split() のところで、組み込み関数の input() で得られた文字列が str クラスのオブジェクトとして生成されて、そのオブジェクトに対して、split()というメソッドが適用されている。
このように、str クラスを定義して、そのクラスに「メソッド」を定義することで、文字列に対して replace や split のような「共通する操作」を定義することができる、というところがオブジェクト指向プログラミングの特徴である。

Python で変数に代入される値は、基本的にはどのような値も「オブジェクト」であり、つまりなんらかの「クラス」を元に生成された「インスタンス」である。そして、ある変数がどのようなクラスから生成されたインスタンスであるかは、type を使って確認することができる。

★次のプログラムを実行してみましょう。

```
s = "Hello"
print(type(s))
i = 1
print(type(i))
n = 1.5
print(type(n))
list = [1,2,3]
print(type(list))
b = True
print(type(b))
```

【クラスの作成】

Python では、「数」「文字列」「リスト」といった組み込みのクラスだけではなく、クラスを自作することができる。様々な種類のデータをオブジェクトとしてとらえて、そのオブジェクトの性質を「クラス」として抽象化し、そのオブジェクトにはどのような属性があるか、そしてそのオブジェクトに対してはどのような操作ができるか、ということをプログラミングしていくことが、オブジェクト指向プログラミングの基本的な考え方である。

★次のプログラムを実行してみましょう。ファイルは 2 つあります。コードは
https://paiza.io/projects/3a6G1hHwwmP-ms2k6Prsog

point.py

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def up(self):
        self.y += 1
    def down(self):
        self.y -= 1
    def right(self):
        self.x += 1
    def left(self):
        self.x -= 1
    def pos(self):
        return self.x, self.y
```

Main.py

```
import point
a = point.Point(2,3)
```