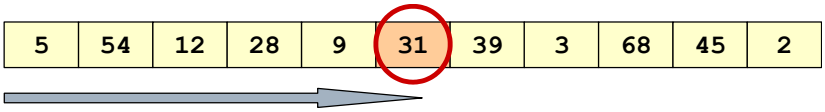
第 11 回　探索アルゴリズム

探索アルゴリズムについて学ぶ。

【線形探索（Linear Search)】

配列の中からある特定の値を持った要素を探すアルゴリズムを、一般に探索アルゴリズムと呼ぶ。探索アルゴリズムには様々なものがあり、もっとも単純な方法が「線形探索」である。線形探索は、配列の先頭から順番に要素を調べていき、探索している数値があるかどうかを調べる。

以下に、線形探索のイメージ図を示す。配列の先頭から順番に調べていき、探索したい値（この例では 31）を探索する。



★線形探索を実現するプログラムを実行しましょう。これまでの知識を活用して「ここではこういう処理をしているのだな」と考えながら打ち込むと良いです。

```
import sys
list = list(map(int,input().split()))
key = int(input())
for i in range(len(list)):
    if list[i] == key:
        print(f"探索成功：配列の {i+1} 番目です。")
        sys.exit()
print("探索に失敗しました。")
```

入力：5 54 12 28 9 31 39 3 68 45 2
　　28
出力：探索成功：配列の 4 番目です。

入力データには、1 行目に探索をするデータの列をスペースで区切って入れて、2 行目には探索する数を入れる。

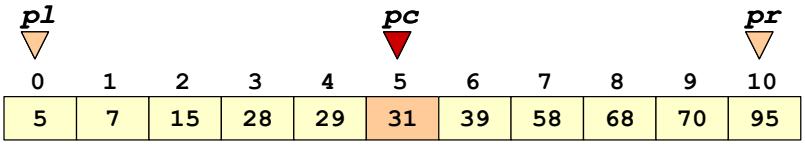
sys.exit() は、プログラムを終了する命令である。sys モジュールの exit 関数を使うので、最初に import sys で sys モジュールを読み込んでいる。このプログラムは、2 つ以上の同じ値がある場合にも、最初の 1 つを見つけた時点で終了する。

【二分探索（Binary Search)】

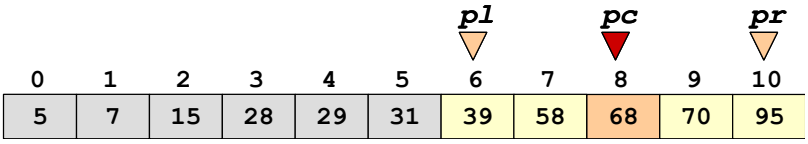
線形探索では，配列の要素の並び順には何も制約がなかった。しかし、これをあらかじめ昇順に整列させておくことで，「二分探索」と呼ばれる、線形探索よりも効率の良い方法で探索できる。たとえば、10 億個のデータから探索する場合には、線形探索では平均で 5 億回、最悪で 10 億回の探索が必要であるが、二分探索であれば、2 の 30 乗が約 10 億なので、10 億個のデータから 30 回以内の探索で目的のデータが見つかる。5 億回と 30 回でどっちが速いかは考えるまでもない。このように、データの数が増えると探索効率の差が歴然とする。

二分探索とは、あらかじめ整列されたデータに対して、探索対象の中央に位置するデータと探索したいデータの大小を比較して探索範囲を半分に絞るという手順を繰り返すことにより、探索したいデータを探すというアルゴリズムである。

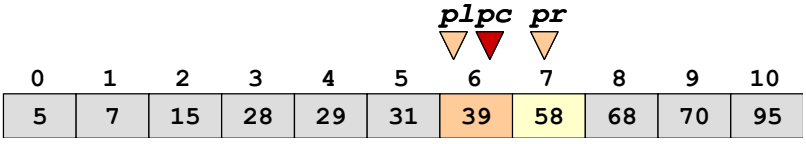
二分探索のイメージ図を以下に示す。二分探索を実行するために、探索範囲の左端を表す変数 pl、探索範囲の右端を表す変数 pr、探索範囲の中央を表す変数 pc を用いる。以下の配列から、39 を探索する場合を考える。まず、pl を左端の 0 番目に、pr を右端の 10 番目に、pc を中央の 5 番目にセットする。



ここで、pc が指し示す 31 と、探索対象の 39 の大小を比較する。探索対象の 39 は 31 より大きいので、31 の右隣から右端までの範囲を新たな探索対象とする。そのために、pl を pc の 1 つ右隣へ移動させ、pc を新 pl と pr の中間にセットする。



ここで、pc が指し示す 68 と、探索対象の 39 の大小を比較する。探索対象の 39 は 68 より小さいので、pl から，68 の 1 つ左隣までを新たな探索対象にする。pc は、pl と pr を足して 2 で割った値を切り捨てて、pl と同じ 6 番目とする。



ここで、pc が指し示す 39 は探索対象に等しいので、探索を終了する。

さて、探索に失敗した場合、何を終了の条件とすればよいだろうか。例えば、上記の例で 40 を探す場合を考えてみよう。pl と pr の位置関係がどうなったときに探索を終了させればよいでしょうか？

【★課題】

二分探索を実現するプログラムの????部分に入る適切なプログラムコードを記述して完成させ、実行して下さい。入力は、小さい数から大きい数へと順番に並んでいる必要があることに注意すること。完成したら、ToyoNet-ACE に提出して下さい。

```
import sys
list = list(map(int,input().split()))
key = int(input())
pl = 0
pr = len(list) - 1
while pl <= pr:
    pc = (pl + pr) // 2
    if ????:
        print(f"探索成功：配列の {pc+1} 番目です。")
        sys.exit()
    elif ????:
        pl = ????
```

```
    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、0 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