## 4.2 素数の表示

pmsg 関数は、入力された数が素数かどうか判定するものだった。それなら初めから素数の一覧表があると便利だろう。そこで素数の一覧を表示するスクリプトを書いてみよう。これは与えられた $n$ までの素数をもれなく表示する。

[py script]

```
>>> def primes(n):
...   pL = [2, 3]
...   for n in range(5, n, 2):
...     for i in pL[1:]:
...       if n % i == 0:
...         break
...     else:
...       pL.extend([n])
...   print(pL)
...
>>> primes(1000)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 22
9, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311,
313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401
, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 4
91, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593,
 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 67
7, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773,
787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881
, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 9
91, 997]
```

スクリプトを説明する前に、素数の調べ方について話しておこう。素数は 2 を除いてすべて奇数だから、3 以上の奇数の中から、約数を持つ数を順にはねれば素数が残る。スクリプトはエラトステネス[1]の篩（ふるい）とは少し違うが、似たような方法で素数を探している。ここでは、3 から順にすべての奇数をテストにかける。テストは、その奇数より小さい素数で順に割ってみるだけである（奇数を割るので素数 2 で割ることはしない）。割れてしまえばその奇数ははねられ、割れなければ素数のリストに追加する、というものだ。スクリプトは `primes(n)` という関数で、`n` までの素数を出力するものである。

では、スクリプトの説明だ。まず、`pL = [2, 3]` に注目してほしい。`[ ]` で囲んだ値を変数に代入すると、それは**リスト**になる。変数名は素数のリスト（primesList）の略であって、リスト用の

[1] エラトステネス（275B.C.–194B.C.）：ギリシア人の数学・天文学者。

名付け規則があるわけではない。**Python3** の変数は、どんな種類の値を代入しようと公平に扱われる。リストに 2 と 3 を代入したのは、素数が 2, 3 から始まることと、割り算テストに使われる最初の素数が 3 だからだ。つまり、この時点でリストには 2 個の素数が格納されていることになる。

3 はリスト入りしているので、次に調べる数は 5 からで、その後 2 ずつ増やした数が調査の候補となる。候補となった数がリストにある素数で割れるかどうか調べるのが、ふたつ目の `for` ブロックである。範囲を示す `in pL[1:]` が新しいね。**Python3** では `for i in pL` と書くだけで、`pL` リストに含まれる数を順に `i` にあてがってくれるのだ。ただし、ここでは 2 で割ることはしないので、`in pL[1:]` としている。これは、リストの 1 番目の要素から最後までを取り出す場合の書き方だ。ちなみにリストの先頭は 0 番目と数えるので、1 番目の要素は 3 である。

もしリストの数で割れれば、`break` 文によって `for` ブロックを抜けて次の候補が調査対象となる。リストの最後まで割り算を試して割り切れなければ、それは素数である。`else` が `if` と同じ字下げ位置ではなく、ふたつ目の `for` と同じ字下げ位置にあることに注意してもらいたい。`else` は `if` 文に付属するものではなく、`for` 文に付属している。すなわちこの `esle` ブロックは、`for` 構文がすべて終わったときに処理されるもので、`break` で抜けるときは実行されない。つまり、素数に対してだけ行われる命令である。

それで `pL.extend([x])` というのは、現在の `pL` リストに `[x]` を追加するものである。こうして最後までリストに追加すれば、リストを表示すればよい。リストの表示なので、`[ ]` で囲まれた値が “, 区切り” で表示される。

ところで、ここに示した `primes` 関数は、計算効率の観点からは大変よくないスクリプトである。それは、たとえば 997 が素数かどうか調べるとき、リストにある 991 までの全部の数で割り算をしている。実際は $\sqrt{991} \approx 31.48$ より小さい素数で割れなければ、それより大きな数で割れっこないのだ。この差は大きい。だから、スクリプトの適切な場所で `i > math.sqrt(n)` を判定して、真ならループを抜ける一文を付け加えるとよい。簡単なことだから、君たちの課題にしておこう。

エラトステネスの篩を用いてスクリプトを書くこともできる。エラトステネスの篩のアルゴリズムは次のとおりだ。

a) $n$ を 2 とする

b) $n$ を素数として確定する（○で囲む）

c) $n$ の倍数をすべて消去する

d) 残っている数で、いちばん小さい数を $n$ とする $\rightarrow$ b) へ戻る

実は、エラトステネスの篩は大変優れたアルゴリズムである。それは、このアルゴリズムで 100 までの素数を調べた一覧を見てもらえば分かる。

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ② | ③ | ~~4~~ | ⑤ | ~~6~~ | ⑦ | ~~8~~ | ~~9~~ | ~~10~~ |
| 11 | ~~12~~ | 13 | ~~14~~ | ~~15~~ | ~~16~~ | 17 | ~~18~~ | 19 | ~~20~~ |
| ~~21~~ | ~~22~~ | 23 | ~~24~~ | ~~25~~ | ~~26~~ | ~~27~~ | ~~28~~ | 29 | ~~30~~ |
| 31 | ~~32~~ | ~~33~~ | ~~34~~ | ~~35~~ | ~~36~~ | 37 | ~~38~~ | ~~39~~ | ~~40~~ |
| 41 | ~~42~~ | 43 | ~~44~~ | ~~45~~ | ~~46~~ | 47 | ~~48~~ | ~~49~~ | ~~50~~ |
| ~~51~~ | ~~52~~ | 53 | ~~54~~ | ~~55~~ | ~~56~~ | ~~57~~ | ~~58~~ | 59 | ~~60~~ |
| 61 | ~~62~~ | ~~63~~ | ~~64~~ | ~~65~~ | ~~66~~ | 67 | ~~68~~ | ~~69~~ | ~~70~~ |
| 71 | ~~72~~ | 73 | ~~74~~ | ~~75~~ | ~~76~~ | ~~77~~ | ~~78~~ | 79 | ~~80~~ |
| ~~81~~ | ~~82~~ | 83 | ~~84~~ | ~~85~~ | ~~86~~ | ~~87~~ | ~~88~~ | 89 | ~~90~~ |
| ~~91~~ | ~~92~~ | ~~93~~ | ~~94~~ | ~~95~~ | ~~96~~ | 97 | ~~98~~ | ~~99~~ | ~~100~~ |

一覧は、7 を素数として確定し、7 の倍数をすべて消したところまでの状況である。次は 11 に○をつけて、11 の倍数を消していく手はずであるが、もうその必要はないのだ。なぜなら、$n$ の倍数を消すことと、消される運命の数を $n$ で割ることは同じことだからだ。すなわち、100 までの素数を調べるための割り算は $\sqrt{100} = 10$ までよい。したがって、アルゴリズムが 11 に到達したところで、100 までの素数がすべて見つかっているのだ（消えずに残った 25 個が素数）。

それなら、これをスクリプトにすれば大変効率的に素数を求められる、と誰もが思うかもしれない。つまり、2 から $n$ までのリストを用意して、2 の倍数から順にリストから除けばよいと考えられるからだ。ところが、リストから数を削除し続けるのは簡単なことではない。削除操作自体は簡単なのだが、削除によって数の順番が詰まってくるために、削除すべき数の位置を特定するのは不可能ではないが、相当に困難だと思われる。

**Python3** にはリストの他に**ディクショナリ**という機能もある。しかし、ディクショナリを用いても、エラトステネスの篩をスクリプトにするには困難が伴うはずだ。結局のところ、アルゴリズムの効率は、必ずしもスクリプトを書く効率と同じではないということだ。