プログラミング入門(Python)

この文書は、プログラムを書いたことがない記者を想定し、プログラムの基本要素を紹介するものです。

なぜ、エンジニアでもないのに、プログラムを書く必要があるのでしょうか？

- 独自の処理
  ソフトウエアが用意していない機能は、自分で書くしかない。 著名なソフトは、マクロ・スクリプト機能を当然のこととして備えている。
- ライブラリの利用
  専門家が公開しているライブラリは、プログラムで制御されることを前提としている。
- デジタルの基本語彙
  プログラムを書けないことは文盲に等しい。（要らぬ敵を作りそう）

プログラムは、プログラミング言語という「言語」で書かれています。これまで、FORTRAN、COBOL、Basic、C、Pascal、C++、Objective-C、Perl、PHP、Python、VisualBasic、Ruby、Java、Javascript、Scala、Haskell、GO、Swiftなど、様々な言語が発明されてきました。

それぞれの言語には「語彙」と「文法」があります。それを使った「コンピューターに対する命令話法」を習得することが、プログラムを覚えることです。外国語を習うことと完全に同じです。

相手がコンピューターなので「理屈しか」通じませんが、学ぶ上では、理屈ではない「エンジニアの慣例」を学ぶことも重要です。買い物した時、「ありがとう」をいうのは売り手なのかか、買い手なのか、辞書や文法書には書いてありません。

日本語で主語が省略できたり、英語で「Cはアルファベットの何番目？」と言えないように、言語には得手不得手があります。概して、高機能な新しい言語ほど（人間にとって）自然に書けますが、実行速度は遅くなります。

北京に住むなら中国語を習得するしかないように、自分の好みで言語を選ぶことはできません。ブラウザで使う場合はJavascript、表計算ソフトならVBAを使うほかありません。

とはいえ、CARのために初めてプログラミングを習得しようと思い立ったのなら、選択肢はPythonの一択です。

> **注釈:**
>
> かつてパソコン少年だった人はこの10年で状況が大きく変わったことに気づくべきです。①メモリ管理が不要な言語が普及し、②使えるメモリが爆発的に増え、③高性能、多様なライブラリがネットで入手でき、④StackOverflowなどでノウハウを検索できるようになりました。コンピュータのために書かなければならなかったコードが大幅に減り、やりたい処理を直接書けるようになりました。これがCoding Journalistが誕生した背景です。

環境設定

Pythonは、バージョン２からバージョン３へ移行し、文法が少し変わりました。ここではpython3を使いますが、python2でも基本は同じです。

まだPython2を使っている人は多く、Macに最初から入っているバージョンはPython2です。Windowsには入っていません。

お勧めの環境はAnaconda(https://www.continuum.io/)のPython3版です。Anacondaは、Python本体だけでなく、numpyなどの有名ライブラリも一括してインストールしてくれます。ただし、サイズは1GBを超えます。

大きなライブラリをインストールしたくない方は、Pythonだけ(https://www.python.org/)をインストールしてください(Windowsの場合)。Macは不要です。ただし、この文書の最後にある「重力波の観測データ」は実行できません。

プログラムの３要素

ターミナルでPython(またはPython3)を打つと、Pythonが起動します。バージョンが表示されますので、確認してください。quit()と入力すると、Pythonは終了します。

以下の命令を打ってみましょう。

```
>>> print(1 + 2)
```

もし「３」が表示されなかった場合、そのコンピューターは壊れているか、コンピューターの形をした別の何かです。

このように、プログラムは「動詞(目的語)」の形で、コンピューターに対して命令をする、その繰り返しです。

多数の命令を一本のテキストファイルにまとめ、一気に実行させることもできます。

テキストファイルsample01.pyを以下のように作成します。

```
print("Hello")
print("Good by")
```

ファイルを作ったら、一度Pythonを終了し(quit())、ターミナルで「python sample01.py」を実行します。

プログラムは「予め書く」という意味ですから、このファイルは立派なプログラムです。

とはいえ、このプログラムは何度実行しても同じことしかしません。状況に応じて気の利いた振る舞いをさせるため、プログラム言語には３つの道具が用意されています。

- 変数代入（Assignment）
- 条件文（Conditional）
- 繰り返し（Iteration）

コンピューターに、コンピューターらしい仕事をさせることができるのは、この３要素のおかげです。

代入(Assignment)

以下のプログラムを入力し、実行させてみましょう。

```
greeting = "Hello"
print(greeting)
```

１行目は、greetingという **変数(variable)** に「Hello」という文字列を代入(assign)しています。

変数は、作者（プログラマー）が名前を自由に決めることができる「データの容れ物」です。ただし、数字で始ったり、途中に+や=などの紛らわしい記号を使うことは許されていません（別の意味で解釈されてしまいます）。

等号記号「=」は、 **数学とは意味が全く違い** 、「右辺のもの(式の値)を左辺の容れ物に入れよ」という命令です。

２行目のprintは、表示命令（関数）ですが、”greeting”という文字を表示するわけではありません。greetingの中身を表示します。

結局のところ、この２行は「print(“Hello”)」と全く同じ意味です。

変数代入がどうしてプログラムに必須な要素なのかは、後でわかります。

演算子(operator)

1+2のような加減乗除の記号を **演算子** といいます。何を意味するか、１行１行、実験してみましょう。

```
print(3 + 2)
print(3 - 2)
print(3 * 2)
print(3 / 2)
print(3 % 2)
# 小数点付き数値の場合
print(3.0 + 2.0)
print(3.0 - 2.0)
print(3.0 * 2.0)
print(3.0 / 2.0)
print(3.0 % 2.0)
# 文字列の場合
print("a" + "b")
print("a" - "b")
# 文字列と数値
print("a" * 3)
```

この結果から分かることは、こんなことではないでしょうか。

- 加算演算子(+)は、整数・小数点付き数値なら足し算、文字列なら連結
- 乗算演算子(-)は、整数・小数点付き数値なら掛け算、文字列と整数なら、その文字列を整数回繰り返した文字列
- 除算演算子(/)は、整数同士なら割ったときの整数部分、小数点付き数値なら本当の割り算
- パーセント演算子(%)は、割り算の余り

このように、演算子は「データの種類」を見て、動作を変えています。以下の例を実行させてみましょう。

```
hensu = 2
print(hensu * 3)
hensu = 2.0
print(hensu * 3)
hensu = "hello"
print(hensu * 3)
```

ここから分かることは、変数は「データの容れ物」になっているだけでなく、その「種類」も知っているということです。

型(type)

データの「種類」を **型(type)** と呼びます。Pythonには型を調べるtypeという命令（関数）が用意されています。

```
type(2)
type(2.0)
type("hello")
hensu = 2.0
type(hensu)
```

それぞれ、int(整数型)クラス、float(小数点型)クラス、str(文字列型)クラスという意味です。クラスは（少なくともpythonでは）型と同じ意味です。

ちなみに、printという関数にも型があります。

```
type(print)
```

builtin_function_or_methodとは「組み込み関数またはメソッド」という型です。

データには型があり、演算子が型に応じて振る舞いを変える 仕組みがあります。

型変換

以下のようなプログラムを実行してみましょう。

```
name = "Taro"
age = 18
print( name + age )
```

表示されるエラーは「型エラー：整数型オブジェクトを文字列型に暗黙的に変換できません」という意味です。

この時、コンピューター側では以下のようなことが起きました。

1. name + ageを評価（計算）した結果を表示(print)しなければならない。しかし、表示できるのは文字列！
2. 文字列型+整数型を計算する場合、加算演算子はできれば文字列を繋ぐ方向で実現させたい（Pythonの方針）
3. もし、後ろの方が文字列型ではない場合は、文字列に勝手に変換して繋ぐ必要がある
4. 整数型を文字列に勝手に変換することは許さない（Pythonの方針）
5. 仕方がないのでエラーを発生させる

つぎのプログラムも実行してみましょう。違いは、表示したいものが逆になっているだけです。

```
name = "Taro"
age = 18
print( age + name )
```

表示されるエラーは「型エラー：整数型+文字列型の演算子はサポートされていません」という意味です。

1. name + ageを表示(print)しなければならない。表示できるのは文字列！
2. 整数型+文字列型を計算する場合、「そんな足し算を試みるなんて気違い沙汰」（Pythonの方針）
3. 「どうせプログラマーの間違えだろ」とエラーを発生させる

ちなみに、こんな例では何が起きているのでしょうか？

```
age = 18
print(age)
```

1. ageを表示(print)しなければならない。表示できるのは文字列！
2. printは中身が文字列型ならそのまま、それ以外なら、そのオブジェクトの__str__メソッドを呼ぶ
3. __str__メソッドが返してきた文字列を表示する

メソッドについては、後で解説します。

このように、printのように文字列型を要求する関数、想定している型が限られる演算子などを使う場合、エラーが発生します。 エラーを除去するのは、プログラマーの仕事です（ **デバッグ** といいます）。最初の場合、整数型を文字列型に変換してやれば、すべて丸く収まります。そのためにstrという関数が用意されています。

```
name = "Taro"
age = 18
print( name + str(age) )
```

はじめてのプログラム

キーボートからの入力を待つ関数にinputがあります。引数には「入力を促す文字列」を指定します。得られるデータは文字列型です。以下のプログラムを試してみましょう。

```
k_data = input("入力してください")
print( "入力したのは「" + k_data + "」ですね" )
```

ここで、初めてのソフトウエア「整数足し算電卓」を作りましょう。骨組みだけ用意しました(sample02.py)。

```
print("a+bを計算します")
a = input("整数aを入力してください")
b = input("整数bを入力してください")
c = a + b
print( "答えは" + c )
```

どう直せはいいでしょうか？

どのプログラミング言語にも、小数型から文字列型から変換したり、文字列型から整数、小数型に変換したりする手段が用意されています。「型変換」はなかなか大変な代物で、マニュアルは手放せません。

条件文（Conditional）

条件文は、コンピューターが「賢くみえる」部分を担当します。つぎの例を見てください。

```
a = 10
if a > 0:
  print("aはプラスの数です")
if a < 0:
  print("aはマイナスの数です")
if a == 0:
  print("aはゼロです")
```

「>」も「<」も演算子で、意味は想像通りです。「==」は「同じかどうか」という意味です。

よく見ると、ある条件が満たされた時だけ実行してほしい命令を、字下げ(indent)して区別しています。これはPython独自の文法です。（javascriptなど、ほとんどの言語は{}を使います）

条件が複合的な場合、AndとOrという単語を使うことができます。これも一目瞭然でしょう。

```
a = 17
b = a % 2   (%演算子は割ったときの余り)
if a > 0 And b == 0:
  print("aはプラスの偶数です")
if a > 0 And b == 1:
  print("aはプラスの奇数です")
if a < 0 And b == 0:
  print("aはマイナスの偶数です")
if a < 0 And b == 1:
  print("aはマイナスの奇数です")
```

コンピューターが賢く見えるのは、上のような **「シナリオ」が予め書かれてある** からです。人工知能も画像解析もメールのスパム判定もすべて同じ理屈です。

もちろん、書くのはプログラマーです。

代入の注意点

代入の「=」は、正確には「右辺の式を評価（計算）した結果を、左辺の変数に代入する」ことです。「評価した結果」が肝です。

```
a = 1
a = a + 1
print( a )
```

「a=a+1」は方程式ではありません。右側を評価して、左側に入れるので、結局2が表示されます。この一見奇妙な表現はプログラムで多用されます。

次の例はどうなるでしょうか。

```
a = 135
isEven = a % 2 == 0
print( isEven )
type( isEven )
if isEven:
  print("偶数です")
else:
  print("奇数です")
```

ブールという型（クラス）はTrueとFalseという値しかとらない「論理値」です。if構文は、正確には「if 論理値:」という構文です。

繰り返し（Iteration）

最後は「繰り返し」です。

その前に、pythonの型であるリスト型(list)を紹介します。下例を実行してみてください。

```
members = ["Taro", "Jiro", "Saburo"]
type(members)
print( member[0] )
print( member[1] )
print( member[2] )
print( member[3] )
```

最初の行の、「[]記号」で囲まれ「,」で区切られたものが **リスト** を表します。（javascriptなどでは配列(Array)と呼ばれているものです）

リストは、[整数]によって「リストの整数番目」にアクセスします。合理的な理由から、pythonでは、リストの整数は0から始まります。（ほとんどの言語も同じです） 最後の行では、「インデックスエラー：範囲外のインデックス」が発生していることに注意してください。

つぎのコードが最小限の繰り返しです。

```
members = ["Taro", "Jiro", "Saburo"]
for m in members:
  print(m)
```

構文「for A in B」は、Bの要素を順に取り出してAに代入し、indentした命令を実行することを、 Bの要素がなくなるまで繰り返します。

繰り返し部分は、それ自体がプログラムなので、 **入れ子構造** が可能です。なお、else(英語で「それ以外」の意味)

は、「直前のifの条件以外の場合」という意味です。

```
kazus = [1,2,3,4,5,6,7,8,9]
for k in kazus:
  a = k * k
  b = a % 2
  if b == 0:
    print( str(a) + "は偶数です")
  else:
    print( str(a) + "は奇数です")
```

さて、1から9までででなく、1000まで繰り返したい場合、1000までのリストkazusを書かなければならないのでしょうか？

pythonには、for構文のために規則的に数字を供給するrangeという特殊な関数が用意されています。

```
kazus = range(1000)
for k in kazus:
  a = k * k
  b = a % 2
  if b == 0:
    print( str(a) + "は偶数です")
  else:
    print( str(a) + "は奇数です")
```

rangeは２つ引数を与えると(初期値、超えたら終了になる境界)と解釈されます。

九九の一覧表は、forの入れ子を使って、以下のように書くことができます(sample03.py)。

```
for n in range(1,10):
  for m in range(1,10):
    seki = n * m
    print( str(n) + "かける" + str(m) + "は" + str(seki) )
```

繰り返しには、forだけでなくwhileという構文もありますが、ここでは省略します。

以上がプログラムの３要素です。

pythonには、下図のように、printやtype、rangeなど78語の「組み込み関数」が用意されています。これは他の言語に比べて多い方ですが、英語の不規則動詞と同じ程度です。

		組み込み関数		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

注釈:

pythonは、オンラインの公式ドキュメント[http://docs.python.jp/3/index.html]があり、チュートリアルも用意されています。ただし、これはプログラミングの心得がある人向けです。英和辞書には大抵のことが書いてありますが、中学１年生には理解不能です。プライドは捨て、入門書をまず通読し、全体像を把握してください。あとは上級者の書いたコードを真似るのみ。外国語学習と同じです。

関数(function)の定義

何度か同じことをする羽目になることがあります。a,b,cの３つの変数があり、それぞれを偶数か奇数か判断しなければならない場合、以下のようになります。

```
a = 135
if ( a % 2 == 0 ):
  print( str(a) + "は偶数です" )
if ( a % 2 == 1 ):
  print( str(a) + "は奇数です" )
b = 63
if ( b % 2 == 0 ):
  print( str(b) + "は偶数です" )
if ( b % 2 == 1 ):
  print( str(b) + "は奇数です" )
c = 82
if ( c % 2 == 0 ):
```

```
  print( str(c) + "は偶数です" )
if ( c % 2 == 1 ):
  print( str(c) + "は奇数です" )
```

下手な記事と同様、冗長というほかありません。 これを避けるため、関数(function)を定義することができます。

```
def GusuOrKisu(n):
  if (n % 2 == 0):
    print( str(n) + "は偶数です" )
    return True
  else:
    print( str(n) + "は奇数です" )
    return False

a = 135
GusuOrKisu(a)
b = 63
GusuOrKisu(b)
c = 82
GusuOrKisu(c)
```

関数と訳されていますが、functionの「機能」の意味です。

関数は、defで始まり、関数名と目的語（引数）を指定して、プログラムをindentして記述します。この場合、変数nが突然登場しますが、驚かないように。関数は以下のように呼び出されます。

- GusuOrKisu(135)によって、GusuOrKisuの行にジャンプする
- 引数の変数に、呼び出し元から渡された数が自動で代入される。つまり、
  n = 135 が自動で実行される。
- 関数内部のプログラムが実行される。
- returnがあれば、値を返すことができる。
  ない場合も、ブロックの最後尾でNone(値なし)という特殊な値が自動で返される。

「関数を呼び出す(call)」「値を返す(return、返値・戻り値)」などの表現は、関数の機能をよく表した表現です。

関数の形式は、pythonの組み込み関数と全く同じです。というより、組み込み関数は「あまりに基本的で汎用性の高い関数だから、pythonがあらかじめ定義しておきましたよ」という心使いです。

上記のプログラムでは、「与えられた数字が偶数か奇数を判定し、表示する」機能が関数GusuOrKisuとしてまとめられ、本体から再利用されていることがよく分かります。

ライブラリ（Library）

組み込み関数ではないものの、pythonには備え付けのライブラリ(モジュール)があります。

例えば、三角関数などは、mathというモジュールに用意されています。sinだけでなく、πも定義されています。ただし、角度は度ではなくラジアン(180度=3.141...)という単位を使います。

```
import math
print( math.sin(45.0 / 180.0 * math.pi) )
```

もし、三角関数のsinを自分で書かなければならないとしたら、ほとんどの人はお手上げです。だから、pythonに限らず、すべてのプログラミング言語は、専門家が書いた高速かつ正確な三角関数を用意しています。

（ただし、盲信は禁物。Javascriptはつい最近まで三角関数が間違っていました）

三角関数には、rangeなどの組み込み関数のように「だれでもいつでも使うほどの汎用性」はありません。そこで、mathという「モジュール」に関数をまとめておき、必要な場合だけ、追加で読み込んでもらうことになっています。冒頭の「import math」は「mathというモジュールを使うので読み込みなさい」という命令です。

用意されているモジュールには、timeやdate(時間を計算する)などがあります。

モジュールは、決められた形式を守りさえすれば誰でも追加できます。

ありがたいことに、統計の専門家は統計専用の、物理学の専門家は物理学専用の、生物学の専門家は生物学専用の「関数コレクション」（モジュール）を作って、公開しています。

この拡張性こそ、pythonを勧める最大の理由です。

中間試験

1785年、8歳のガウスは1から100まで全部足す課題を瞬時にこなし、先生を驚かせたそうです(事実かどうか不明)。1から1785まで、すべての整数を合算してください。

アルゴリズム(algorithm)

素数(prime number)は「自然数で、正の約数が1と自分自身のみであるもの」のことです。ある数が素数かどうか判定する関数を作ってみましょう。

以下の例は、与えられた整数を「2以上n未満の整数で、順次nを割り切れるかどうか調べる」という方針です(sample04.py)。

```
def isPrime(n):
  # 2以上n未満で割り切れたらFalseを返す関数
  for i in range(2, n):
    if n % i == 0:
      return False
  return True

a = 135
if isPrime(a):
  print( str(a) + "は素数です" )
else:
  print( str(a) + "は素数ではありません" )
```

素数14414443でも試してみましょう。あなたのパソコンの能力が分かります。

アルゴリズムとは、「2以上n未満の整数でnが割り切れるかどうかを順次試す」というような **計算方針** のことです。アルゴリズムは、プログラムの性能を決定的に左右します。そして、パソコンではなく、プログラマの能力が分かります。

上例のisPrimeのアルゴリズムを改善してみましょう。

中級：クラス(class)

高級言語には、最初から組み込まれた型（組み込み型）と同じように振る舞うクラス(class)が用意されています。

以下のプログラムは、辺の長さが3、4、5である三角形と、辺の長さが6、5である長方形の面積を表示します。三角形と長方形では、面積を求める公式は違うにも関わらず、とても自己説明的です。

（プログラム全体はsample05.py）

```
obj1 = Triangle(3,4,5)
obj2 = Rectangle(6,5)
print( "obj1の面積は" + str( obj1.getArea() ) )
print( "obj2の面積は" + str( obj2.getArea() ) )
```

クラスは、プログラムを **このように書くための仕組み** です。発想が英語話者です。

事前に、Triangleというクラスと Rectangleというクラスを **定義** しておきます。

```
import math
class Triangle:
  def __init__(self,a,b,c):
    self.a = a
    self.b = b
    self.c = c
  def getArea(self):
    # ヘロンの公式
    s = (self.a + self.b + self.c) / 2.0
    return math.sqrt( s * (s - self.a) * (s - self.b) * (s - self.c) )
class Rectangle:
  def __init__(self,a,b):
    self.a = a
    self.b = b
  def getArea(self):
    return self.a * self.b
```

クラスTraiangleは、３辺の長さをa、b、cとして保持し、getAreaというメソッド(クラス専用関数)を備えています。同様に、クラスRectangleには２辺とメソッドgetAreaがあります。

両方ともgetAreaというメソッドを持ち、その実質（計算部分）は違います。その結果、プログラムを自己説明的に書いても、正しい計算方法を選んでくれます。

このプログラムには、今後の修正で、五角形や円も必要になるかもしれません。クラスを使うことで、本体部分を大きく書き換えることなく、修正できます。

中級：関数オブジェクト

（この項は少し高度な内容なので、頭の片隅に入れるだけで結構です）

PythonやJavascriptでは、関数も、変数と同じ「オブジェクト」として扱われます。つまり、数字の135や文字列の”Hello”と同じように、代入したり、関数の引数にしたり、あたかもモノのように扱うことができます。

以下の例は、「２乗を表示する」という関数printSqrtを変数jobに代入し、mapという組み込み関数を使って実行させるものです。map関数は、関数オブジェクトとリストを引数にとり、リストの各要素に関数を適用した結果を返します。(sample06.py)

```
def printSqrt(n):
  print( n * n )
  return n * n

job = printSqrt
items = [3, 6, 2, -1]
res = map( printSqrt, items )
print( list(res) )  # 遅延実行される
```

mapを使わなければならない場面は多くありませんが、上級者が書いたプログラムを参考にするときには知っておく

必要があります。上級者がmapを使うのは、処理そのものをモノとして扱うことで、プログラムの意図が分かりやすくなると信じているからです。

重力波の観測データ

2015年9月14日9時50分45秒に初めて観測された重力波のデータは、観測グループLIGOによって、 **正式発表の前から**公表されていました。

[https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.html]

（GW150914_tutorial.pyを実行してみること）