仮入部員向け競プロ講習会
Beginner編

KARASU4280, initial_d, maccha

Day1 2020年6月20日
Day2 2020年6月27日

目 次

第 I 部
はじめに

1　競プロとは

競技プログラミング (以下、競プロ) とは、名前の通りプログラミングを用いた競技である。詳細を、Wikipedia[1] から引用する。

> 競技プログラミングでは、参加者全員に同一の課題が出題され、より早く与えられた要求を満足するプログラムを正確に記述することを競う。コンピュータサイエンスや数学の知識を必要とする問題が多い。多くのコンテストでオンラインジャッジが採用されている。
>
> また、競技プログラミングに参加する人を「競プロ er」と呼ぶことがある。

1.1　魅力

我々競プロ er が競プロに取り組む理由は、次の通りである [3]。

1. プログラミングスキル向上
2. アルゴリズムの習得
3. 数学的考察力の向上
4. 大人数とリアルタイムで競える

これらが、代表的な魅力だ。

幣学の学生の多くは、プログラミングに興味ある者が多いように思われる。電気通信大学では、1 年次の後期の必修科目で「Ruby」と「C 言語」を学ぶ。また、今年度から前期の「総合コミュニケーション科学」にて、「Python」にも触れる。プログラミングに触れる機会は必ず訪れる。しかし、講義だけでモチベーションは維持できるだろうか。演習量は、足りるだろうか。プログラミングの学習がより楽しいものになれば、モチベーションがもっと上がるはずだ。競プロの最大の魅力は、自分の書いたコードで、ライバルと競い合う「ゲーム」性にあると思う。このゲーム性が一番面白いといっても過言ではない。

競プロで競うポイントを挙げる。

1. 正確か正確でないか
2. 解答速度
3. 実行時間
4. メモリの使用量

以上の点を、同じ言語同士、あるいは異なる言語で同士で競い合う。

正確か正確でないかは、正しいかどうかということだ。試験での丸かバツかということでもある。解答速度とは、解き始めてから解き終わる (解答コードを提出) までの所要時間だ。ここが競プロで重要な評価基準である。実行時間とメモリの使用量については、コンテストでは一部の例外を除いて、評価対象ではない。

1.2 AtCoder における実力評価

近年のコンテスト参加者は国内外で、増加傾向にある。世界最高峰の競技プログラミングサイト「AtCoder」では、コンテストにおける実力を次のように評価している。情報元は AtCoder 社の代表である chokudai 氏のブログだ [2]。

灰 参加すれば誰でもなれるので意欲以外の保証はなし。

茶 学生で茶色なら優秀だがエンジニアとしてはちょっと物足りない。派遣とかで来たエンジニアが茶色あれば一安心。

緑 大抵の企業でアルゴリズム力は十分。AtCoder 的には決して上位ではないが、他社評価サイトなら最高評価。

水 基礎的なアルゴリズム処理能力については疑いのないレベル。

青 青以上は一部上場の IT 企業でも、一人もいないことが結構あるレベルになる。

黄 黄色からは化け物。競プロの問題を解く機械だと思っておけば良い。

橙 あたまおかしい。

赤 もうなんか世界大会とかに招待されたりする。

Color	Rank	Rating	上位%
赤	SSS	2800 - 3199	0.3
橙	SS	2400 - 2799	1
黄	S	2000 - 2399	3
青	A	1600 - 1999	7
水	B	1200 - 1599	15
緑	C	800 - 1199	30
茶	D	400 - 799	50
灰	E	1 - 399	100

2 AtCoderの始め方

1. アカウント作成
2. 言語を決める。こだわりがないなら、「C++」を勧める。
3. C++を使うなら、『C++入門 AtCoder Programming Guide for beginners (APG4b)』(`https://atcoder.jp/contests/APG4b`) でプログラミングを学習
4. 『AtCoder Beginners Selection』(`https://atcoder.jp/contests/abs`) を解く
5. 『AtCoder（競技プログラミング）の色・ランクと実力評価、問題例』(`http://chokudai.hatenablog.com/entry/2019/02/11/155904`) をさらっと読む。完璧な理解は不要。
6. 過去の「AtCoder Beginner Contest」の A, B, C 問題を解く。
7. 「AtCoder Beginner Contest」に参加

「C++」を勧めるのは、AtCoderでの利用者が大多数であることと、その実行速度の速さである。ネットで調べる際も、「C++」の記事が目立つ。既に経験のある言語がないなら、お勧めする。

『C++入門 AtCoder Programming Guide for beginners (APG4b)』でプログラミングを学習とあるが、一度で完璧にマスターする必要はない。困った時や時間が経ってから、再度復習するつもりで、さらっと済ませても良いと思われる。

『AtCoder Beginners Selection』は、直ぐにとは言わないが、全部解けるようになって欲しい問題だ。APG4bの2章が済んだら、解ける問題からやり始めると良いだろう。

『AtCoder（競技プログラミング）の色・ランクと実力評価、問題例』を読むと、コンテスト及びAtCoderの、実力評価の指標であるRatingについて分かる。完璧な理解は必要なく、簡単に読んで頂けたら充分だ。

何よりもRatingを上げるには、コンテストに参加するのが一番だ。「AtCoder Beginner Contest」に積極的に参加しよう。初めは、A, B問題を完答することを目標にしよう。加えて過去のコンテストの問題を時間があるときに解いておくとよい。上級者ほど、過去問を解いている人は多く、新しく入った方でも、過去問を解くことでみるみる実力が上がっていくので、ぜひ取り組んでほしい。

また、AtCoderで精進するに当たって、「AtCoder Problems」の利用を強く薦めたい。これは、有志が運営する、AtCoderのための神サイトである。AtCoderの他に、GitHubのアカウントが必要である。自分の問題の解答状況のほか、バーチャルコンテストの開催、参加ができる。

3 講習会の進め方

1回目は、C++とRubyの紹介をする。2つの言語を体感し、自分に合いそうな言語を探してもらいたい。既に自分の言語が決まっている方は、その言語でも差し支えない。講習内容が既知であったら、次の問題を解いておくことを勧める。

2回目では、1回目で学んだこと事項で、実際に過去問を解いていきます。

参考文献

[1] Wikipedia, 『競技プログラミング』(https://ja.wikipedia.org/wiki/競技プログラミング), 最終閲覧日 2020/6/20

[2] chokudai(id:chokudai), 『AtCoder（競技プログラミング）の色・ランクと実力評価、問題例』(http://chokudai.hatenablog.com/entry/2019/02/11/155904), 最終閲覧日 2020/6/20

[3] E869120, 『レッドコーダーが教える、競プロ・AtCoder 上達のガイドライン【初級編：競プロを始めよう』(https://qiita.com/e869120/items/f1c6f98364d1443148b3), 最終閲覧日 2020/6/20

第II部
付録 Python

maccha

1　Pythonについて

今回の競プロ講習会では、Pythonを教えることはない。この資料は付録として、Pythonの紹介をしたい。競プロで、提出率第2位を誇る言語である。

1991年に誕生したプログラミング言語。静的型付けのC言語とは違った、動的型付けという特性をもつ。機械学習など、近年注目を浴びている言語。Python3の台頭によりPython2からの移行が進みつつある。ここでWikipedia[2]からの引用を掲載する。

> Pythonは、汎用のプログラミング言語である。コードがシンプルで扱いやすく設計されており、C言語などに比べて、さまざまなプログラムを分かりやすく、少ないコード行数で書けるといった特徴がある。

AtCoderでは、Pythonで書かれたコードの提出数が2位と、メージャーである。便利なパッケージが揃っており、また誰が書いても似たようなコードになるように設計されているので、競プロでも人気な言語だ。

1.1　Pythonの特徴

- 改行時に「;(セミコロン)」は必要ない
- 変数を使う前に型を宣言する必要がない
- プログラミングの終了に、「return 0」が不要

変数や関数の返り値などの値の型が、プログラム実行時に決まる性質を動的型付けという。対して、C言語のように型宣言をする性質を静的型付けという。ただし、Pythonにおいて、型が存在しないということではない。Pythonの型は、プログラム実行時にコンパイラが判定、推測してくれる。そのため、静的型付け言語と比べて、実行時のコンパイラの作業が多いため、実行速度とメモリ使用量に劣る。

2　型

Pythonでは、小数はfloatのみである。また、int型とfloat型との演算ができる。演算結果はfloat型となる。前述の通り、Pythonの変数の型は実行時にコンパイラが判定、推測する。

Listing 1: Python_type

```
i = 1 # int
f = 1.0 # float
g = float(i) # from int to float
j = int(f) # from float to int
s = "AtCoder" # str
```

1, 2, 5 行目のコードから、コンパイラは各変数の型を推測する。コンパイラは、代入したい情報が「””」で囲まれいれば str 型 (文字列)、そうではなく小数点「.」を含むなら float 型 (小数型)、それ以外 (整数としての表記) なら int 型 (整数型) と判定する。型を変更したいなら、組み込み関数[1]である int, float, str 関数を使う。

意図しない型に判定されないよう、型が決まることになる初期の宣言 (初期化) では、注意が必要である。

3 四則演算

四則演算のルールは確認したい。

Listing 2: Python_calc

```
a = 3
b = 2

c = a + b # c is 5.
c = a - b # c is 1.
c = a * b # c is 6.
c = a / b # c is 1.5.
c = a // b # c is 1.
c = a % b # b is 1.
```

以降は、コメントとして、実行時には無視される。注意したいのは割り算である。

Python では、float 型と int 型との四則演算も可能だ。また、四則演算の記述方法は、C 言語などの他言語と共通するが、割り算の商を得るには、演算子「//」を用いる。

4 入力と出力

入力 t と出力ができなければ、競プロは始まらない。

Listing 3: Python_inout

```
a = input()

print("Hello", "World", end = "")
print(a)
# Hello World "value of a"
```

[1]言語の仕様として、標準で使用できる関数。これに対して、3, 4 行目のように、ユーザ自身がコード上で定義する関数を「ユーザ定義関数」という。

「input」関数は、入力された文字列 (一行) を返す[2]組み込み関数である。返り値が str 型であることに留意すること。ここでは、変数 a に入力された入力の 1 行目が代入される。そして、変数 a は str 型である。

出力には、「print」関数を用いる。指定した文字列を出力するには、「””」でその文字列を囲み、print 関数に渡す。print 関数に渡す値 (引数) が複数あるならば、「,」で区切らなければならない。また、変数の値も出力できる。このとき、その変数は「””」で囲まない。つまり、コンパイラは print 関数の引数が「””」で囲まれているか否かで変数かどうかを判定しているのだ。

print 関数は、出力の最後に改行が行われる。もしも、改行してほしくない時には、print 関数の引数の最後に「end = ””」を加えるだけでよい。

5 文字列 (str)

Python における文字列の型は str 型のみである。

Listing 4: Python_str

```
s1 = "At"
s2 = "Coder"

s12 = s1 + s2

print(s12) # AtCoder
print(s12[0], s12[2])
print(s12[2:7])
```

文字列を「+」で結合することも可能だ。変数 s12 には、”AtCoder” という文字列が格納される。

また、文字列の一部分を出力することも可能だ。文字列 s の先頭を 0 番にすれば、i 番目の文字はコード上で次のように表現する。

```
s[i]
```

さらに、i 番目から j 番目までの文字列は次のように表現する。

```
s[i: j + 1]
```

6 if

下記のコードは、入力で得た整数 n が正なら”Positive”、0 なら”Zero”、それ以外 (負) なら”Negative” と出力する Python のプログラム。

Listing 5: Python_if

```
n = int(input())
if n > 0:
  print("Positive")
elif n == 0:
  print("Zero")
```

[2]関数などを処理した結果として得られる値を「戻り値」や「返り値」といい、この動作を「返す」と表現する。

```
else:
  print("Negative")
```

一方、C++だとこうなる。

Listing 6: C++_if

```
int main(){
  int n;
  cin >> n;
  if(n > 0){
    cout << "Positive" << endl;
  } else if(n == 0){
    cout << "Zero" << endl;
  } else {
    cout << "Negative" << endl;
  }
}
```

int 関数の引数が小数のときは、引数である数値以下の、最大の整数を返す。例えば、小数 33.4 を引数として渡せば、整数 33 が返される。

では、if 文ついて説明する。C++と Python の違いを述べる。

- 条件式を”()”で囲まない
- 条件式の末尾に”:”におく
- C++でいうブロックの中身 ({} の中身) は、直属の if 文から 1 回インデントする必要がある
- ”&&”ではなく、”and”
- ”||”ではなく、”or”
- ”else if”ではなく、”elif”

7 for

for は、(カウンタ変数が) 条件を満たす間、ブロック内の処理を繰り返す。ある for に属する処理は、所属先の for 文から 1 つインデントして書く。下記のコードは、0 から 9 までの整数を順に出力するプログラムである。

Listing 7: Python_for1

```
for i in range(10):
  print(i) # 0 1 2 3 4 5 6 7 8 9
```

range 関数はカウンタ変数 i に数値を与える役割を持つ。for のブロック内の処理を 1 周すると、カウンタ変数 i の数値に 1 を加える。range 関数の引数が 1 つだけのとき、カウンタ変数 i は、for のブロック内の処理を 1 周するごとに、「i = 0」(初期値) から 1 ずつ増えていき、「i = 9」まで繰り返す。「i = 9」のときの処理が終わると、「i = 10」(終了値) となり、直ちに for 内の処理は実行されなくなる。

カウンタ変数の初期値を指定したい時の例を示す。

Listing 8: Python_for2

```
for i in range(1, 10):
  print(i) # 1 2 3 4 5 6 7 8 9
```

終了値より先に、引数として指定する。カウンタ変数は、初期値「i = 1」から処理 1 周ごとに「i = 2, 3, . . ., 9, 10」と増えてゆき、「i = 10」となった時点で処理は終了する。この引数を省略すると、初期値は「i = 0」となる。

さらに、カウンタ変数を好きな数ずつ増やす方法を紹介する。次のように記述する。

Listing 9: Python_for3

```
for i in range(1, 10, 2):
  print(i) # 1 3 5 7 9

for i in range(0, 10, 2):
  print(i) # 2 4 6 8
```

1 つ目の for 文では、初期値「i = 1」から処理 1 周ごとに「i = 3, 5, . . ., 9, 11」と増えてゆき、「i = 11 (¿= 10)」となった時点で処理は終了する。この引数を省略すると、カウンタ変数は 1 ずつ増えていく。

2 つ目の for 文では、初期値「i = 0」から処理 1 周ごとに「i = 2, 4, . . ., 8, 10」と増えてゆき、「i = 10」となった時点で処理は終了する。

Listing 10: Python_for4

```
range(start, stop[, step])
```

start 引数 start(初期値) の値 (この引数が与えられていない場合は、0)

stop 引数 stop(終了値) の値

step 引数 step の値 (この引数が与えられていない場合は、1)

8 list

Python では、配列のことを「list」と呼ぶ。list の宣言方法は次の通り。非常にシンプルである。

Listing 11: Python_list1

```
l = [] # l is []

l = [0, 1, 2, 3] # l is [0, 1, 2, 3]
print(l)  #[0, 1, 2, 3]
print(l[2]) # 2
```

list の i 番目の要素を取り出すには、

```
l[i]
```

と記述する。

Listing 12: Python_list2

```
l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(l[1:5]) # [1, 2, 3, 4]

print(l[6:]) # [6, 7, 8, 9]

print(l[:3]) # [0, 1, 2]
```

list の一部を取り出すには、上記のような記述をする。返ってくるのは、要素ではなく、list であることに注意せよ。記述方法は、range 関数と似ている。$i <= j$ として、list の i 番目から j 番目までを取り出すには、

```
l[i:j+1]
```

と記述する。

(j+1) の部分がない場合は、i 番目から一番後ろの要素までの list を取り出す。

```
l[i:]
```

一方、i を省略すると、一番前の要素から j 番目までの list を取り出す。

```
l[:j+1]
```

Listing 13: Python_list3

```
l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(l[1:6:2]) # [1, 3, 5]
print(l[1:7:2]) # [1, 3, 5]
print(l[1:8:2]) # [1, 3, 5, 7]
```

list の i 番目から j 番目までを k 個おきに取り出すとき、

```
l[i:j+1:k+1]
```

と記述する。

参考文献

[1] ©AtCoder, 『AtCoder：競技プログラミングコンテストを開催する国内最大のサイト』(https://atcoder.jp), 最終閲覧日 2020/6/7

[2] Wikipedia, 『Python』(https://ja.wikipedia.org/wiki/Python), 最終閲覧日 2020/6/7

[3] Python Software Foundation, 『Python 3.8.3 ドキュメント』(https://docs.python.org/ja/3/), 最終閲覧日 2020/6/7

[4] Al Sweigart 著, 相川愛三 訳, 『退屈なことは Python にやらせよう - ノンプログラマーにもできる自動化処理プログラミング』, オライリー・ジャパン, 2019