第３３回世界コンピュータ将棋選手権

きふわらべ
アピール文書
（大会後差替）

２０２３年０５月０７日　高橋智史

SANSYO

３勝　したぜ

JIKO

実質的に、自己ハイ

22	拡張	7×	21×	15×	3×	27×	17×	9×	23×	0	8	0	26.0	0.0	0.0	28
23	きふわらべ	6×	20×	18○	8×	16×	27○	19×	22○	3	5	0	21.0	4.0	1.0	25
24	HoneyWaffle	5○	19○	3○	12×	1○	2○	28×	7○	6	2	0	41.0	27.0	18.0	4

開発者
高橋　智史　（著作権上、問題ありません）

「 きふわるべ（※１）　と、　きふわらべ軍団（※２）　を除くと
自己ハイの３勝だぜ。　３勝５敗」

※１　…　2016年5月　第２６回世界コンピュータ将棋選手権（ＷＣＳＣ２６）
に参加した、きふわらべの通称。
わたしの中での識別名称は　きふわらぷりー。
４勝３敗。「大樹の枝」ライブラリーを使用。

※２　…　2020年10月　第１回世界将棋ＡＩ電竜戦に参加した、
複数のきふわらべの通称。
わたしの中での呼称は　きふわらべーず。　９体で参加して
兄弟を倒して星を増やした。　一番強いきふわらべは　３勝７敗。

コンピュータ将棋エンジン
きふわらべ　（著作権上、問題ありません）

「 ３回戦の　重力場計算法　は　１手詰めを見逃してくれて、
６回戦の　ＪＨＢＲ　は　こっちが王手したら　投了してくれたぜ」

フロム・スクラッチ宣言
思考部に大きな影響を与える、他者の作成したプログラム・データ等を利用していません。

フリーフォント「たぬき油性マジック」　作者：たぬき侍　（利用ライセンス上、問題ありません）
https://tanukifont.com/tanuki-permanent-marker/

ひよ子　（著作権上、問題ありません）

「　８回戦の　臥龍　には　頭金を決めてるわよ」

「きふわらべが　将棋のルールで　まともに勝ったのは
ＷＣＳＣ３１の　ponkotsu　戦に続いて　２例目だぜ」

「わたしは　持ち駒を打って　利きを当てれば　駒を取れると思ってるから、
それが　たまたま王手で、　飛車も利いていたのが　ラッキーだったぜ」

HEIRETU

わたしは　並列　処理を初めて使ってみた

SHI-PI-YU-

並列処理は、複数の　ＣＰＵ　を同時に使うこと

第２９回世界コンピュータ将棋選手権　きふわらべ　アピール文書
https://www.apply.computer-shogi.org/wcsc29/appeal/Kifuwarabe/appeal.pdf

「 ☞　４年前の　ＷＣＳＣ２９　出場時から
Ryzen Threadripper 2920X
12コア　を持ってるんだが、

パラレル・プロセッシング
（Parallel processing；並列処理）
のプログラムの書き方を　知らなかった」

「 新しいものは　買っただけで
満足するタイプですからね」

「 ☟　エリクサー（Elixir）言語が
パラレル・プロセッシングを
得意としてて、
Flow ライブラリーを使うだけで
いいそうだぜ」


```
28      move_list_on_board =
29        if remain == 1 or remain == 2 do
30          #
31          # 盤上の自駒
32          # ＝＝＝＝＝＝
33          #
34          # ▷ スペース（:sp；空マス）は除去。かつ、手番の駒だけ残す
35          # ▷ （並列化） `Flow.from_enumerable()` - コンピューターの各コアへ処理を振り分け
36          # ▷ （並列化） ピース（Piece；先後付きの駒種類）から、先後を消し、ピース・タイプ（Piece Type；駒種類）に変換する
37          #              マス番地と駒種類から、指し手生成
38          # ▷ （並列化終り） `Enum.to_list()` - 振り分けた処理をリストに戻す
39          # ▷ リストがネストしていたら、フラットにする
40          # ▷ 指し手が nil なら除去
41          move_list_on_board =
42            pos.board
43            ▷ Enum.filter(fn {_sq, piece} →
44              piece ≠ :sp and pos.turn == KifuwarabeWcsc33.CLI.Mappings.ToTurn.from_piece(piece)
45            end)
46            ▷ Flow.from_enumerable()
47            ▷ Flow.map(fn {sq, piece} →
48              piece_type = KifuwarabeWcsc33.CLI.Mappings.ToPieceType.from_piece(piece)
49              pos ▷ make_move_list_by_piece_on_board(sq, piece_type)
50            end)
51            ▷ Enum.to_list()
52            ▷ List.flatten()
53            ▷ Enum.filter(fn move → !is_nil(move) end)
54
55          move_list_on_board
56        else
```

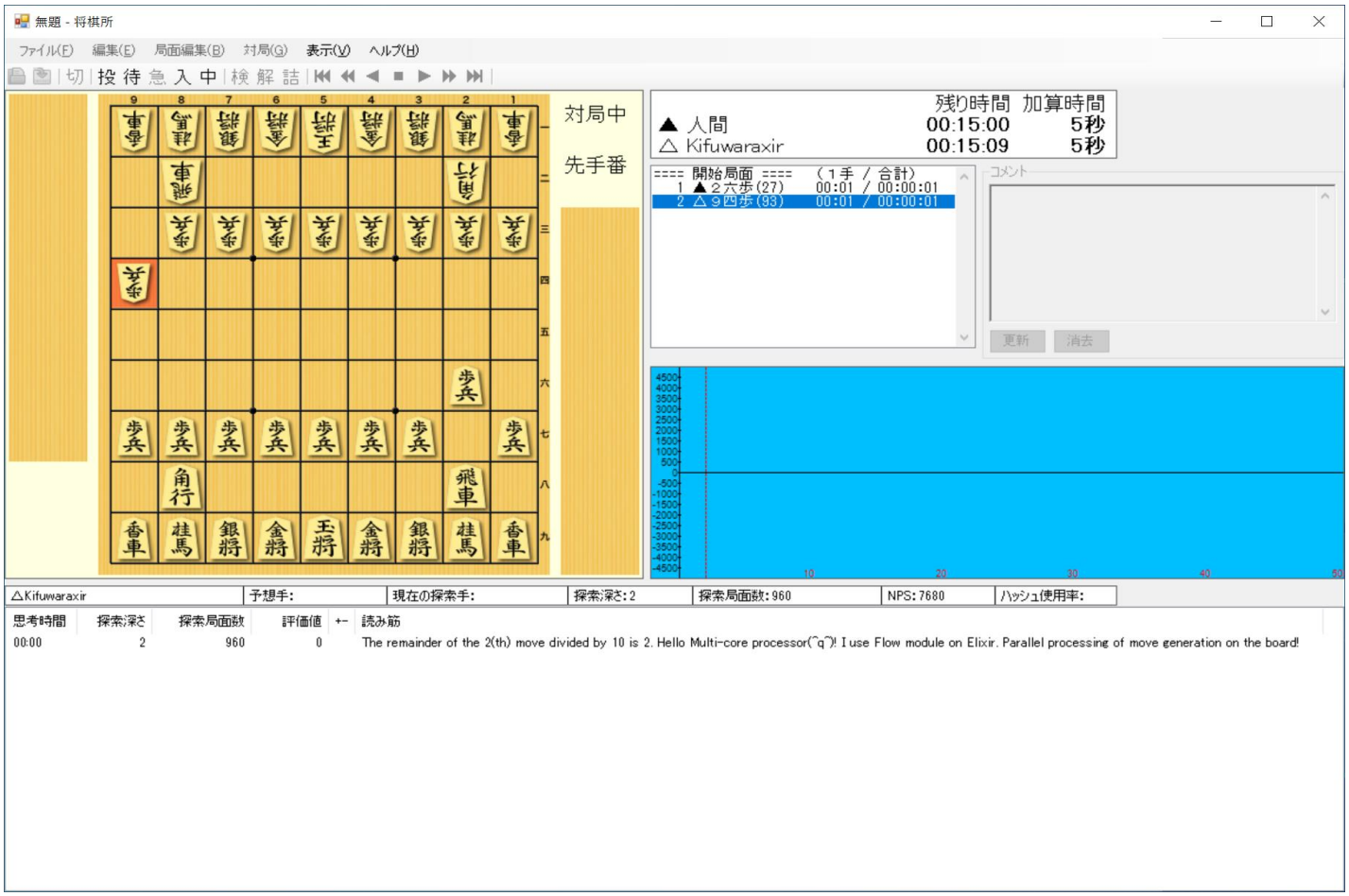
「 初めて　パラレル・プロセッシング　を使ってみて
どうだったんだぜ？」

「 低速化した。
２手読み　０秒　だったところ、１分以上かかるようになった。

嫌になって、パラレル・プロセッシングする回数を　減らすことにしたぜ」

◆ 今何手目かを１０で割って　余りが　1、2　のときは
盤上の駒を動かす指し手　を並列処理で生成する

◆ 今何手目かを１０で割って　余りが　3、4　のときは
持ち駒を打つ手　を並列処理で生成する



「 ☝ きふわらべちゃん、なんか　読み筋の欄を使って　何か言ってるわねえ」

「盤上の駒を動かす指し手を　パラレル・プロセッシングして生成したと言ってるぜ。

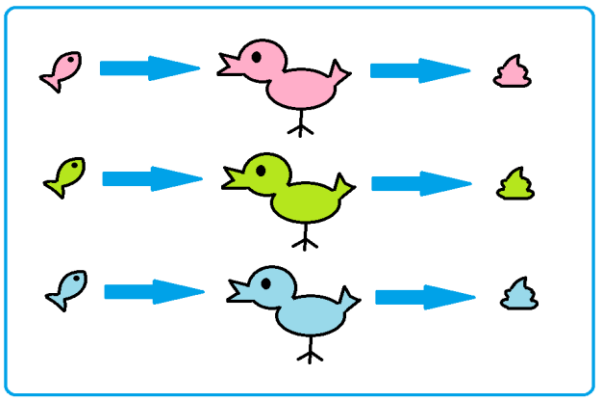
ＷＣＳＣの大会サーバーは　読み筋以外のコメントは受け付けないので公式の棋譜には残っていないぜ」

「今まで使ってなかった　スレッドリッパーのマルチコアを使ってパラレル・プロセッシングできて　満足だぜ」

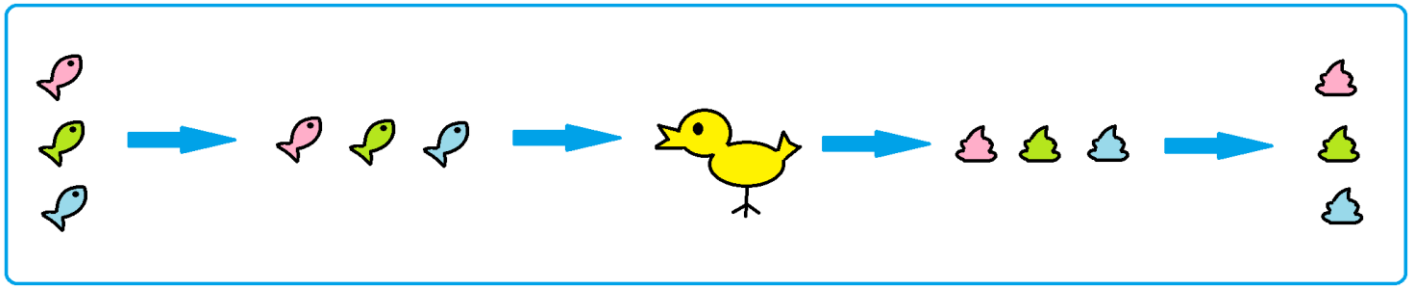
「お父んが　ＷＣＳＣ３１、３２　で使っていた　ゴ（Go）言語　には

Flowモジュールなんかインストールしなくても　最初から
コンカレント・プロセッシング（Concurrent processing；並行処理）　のゴルーチン（Goroutine）　が付いてるそうだぜ」

「パラレル・プロセッシング（Parallel processing；並列処理）と、
コンカレント・プロセッシング（Concurrent processing；並行処理）は違うんで」



Parallel processing



Concurrent processing

GI－PI－YU－

わたしは　ＧＰＵ　を初めて計算に使った気分になった

GI－PI－GI－PI－YU－

ＧＰＧＰＵ　は、　ＧＰＵ　を計算に使うこと

「☞　新しい　ＧＰＵ　を
買ったぜ」

「 GeForce RTX 4090　ね」

「 GeForce RTX 3080 Ti を既に持ってたのに　なんでまた上位モデルを買う？
ダブってるじゃないかだぜ？」

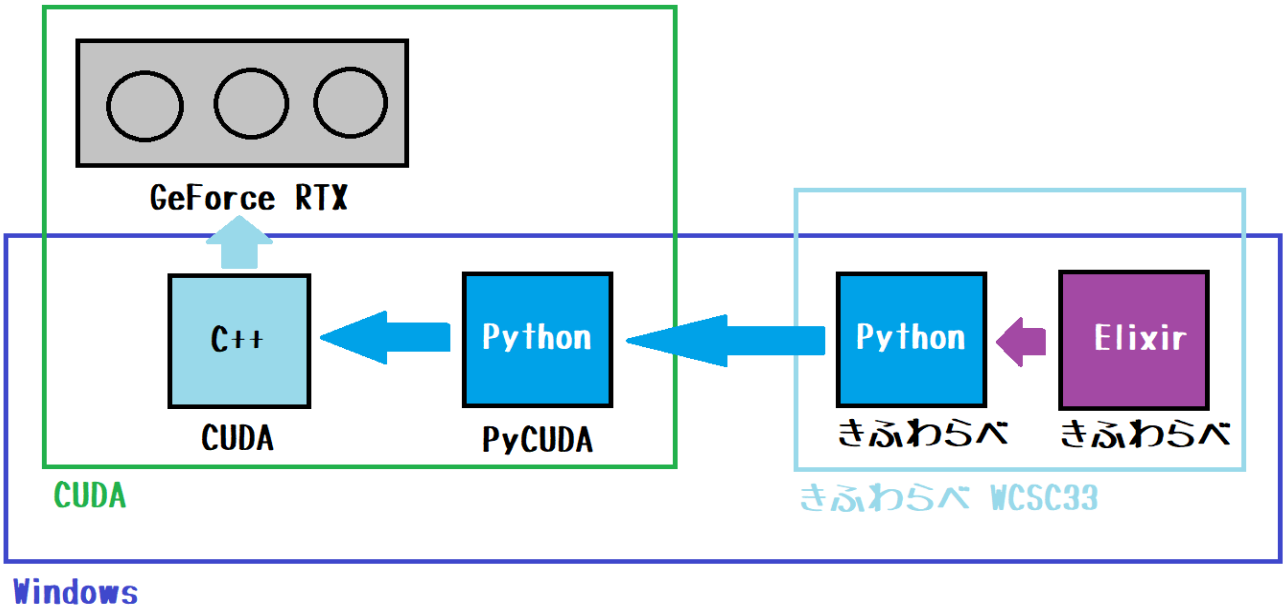
「 わたしは　ヒャッホー　と思えるものを　気軽に　買う。
人生の良い選択とか　決意とか　経済合理性のようなものは　無い」

「 そうでしょうね」

「 わたしは　お父んが　ヒャッホー　したいから作られたＡＩだしな」

「 パイトーチ（PyTorch）で 機械学習のエグザンプルを　動かせば
ＧＰＵの使い方を知らなくても　ＧＰＵは使われるわけだが……」

「 下がっていく数値や、折れ線グラフを見ても
ＧＰＵを使っている気分がしない、というのが
わたしに　合わなかった。
そこで今回、その気分を　上げることにした」



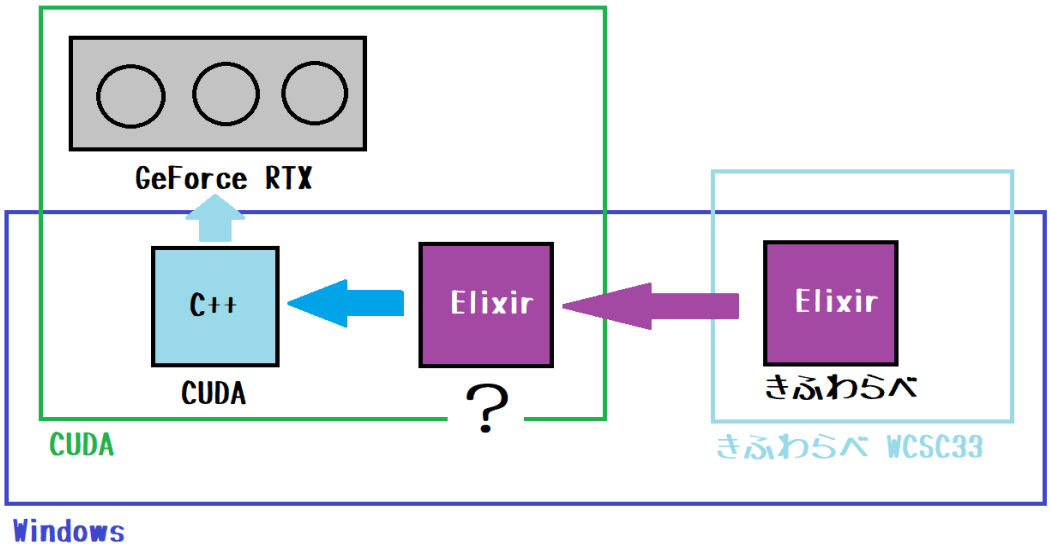
「 ☝　エヌビディア（NVIDIA）という会社が作っている
ジーフォース（GeForce）というイケてるグラフィックボードを使うには、
クーダ（CUDA）というライブラリを　プログラムから使えばいいんだが」

「 クーダは　シープラプラ（Ｃ＋＋）言語で書かれていて、
きふわらべは　エリクサー（Elixir）言語で書かれていて
言語が違って　直接使うことは　できないのだった」

「 クーダを直接使ったら　気分が上がるのね」

「シープラプラ　はなんか　難しかったので、
パイソン（Python）言語で書かれているパイ・クーダ（PyCUDA）ライブラリ
に間に入ってもらうことにしたんだぜ」

「じゃあ　エリクサーのパラレル・プロセッシングは
ＧＰＧＰＵには　利用されてなくて、
パイソン　使ってるのと　同じなんだ」



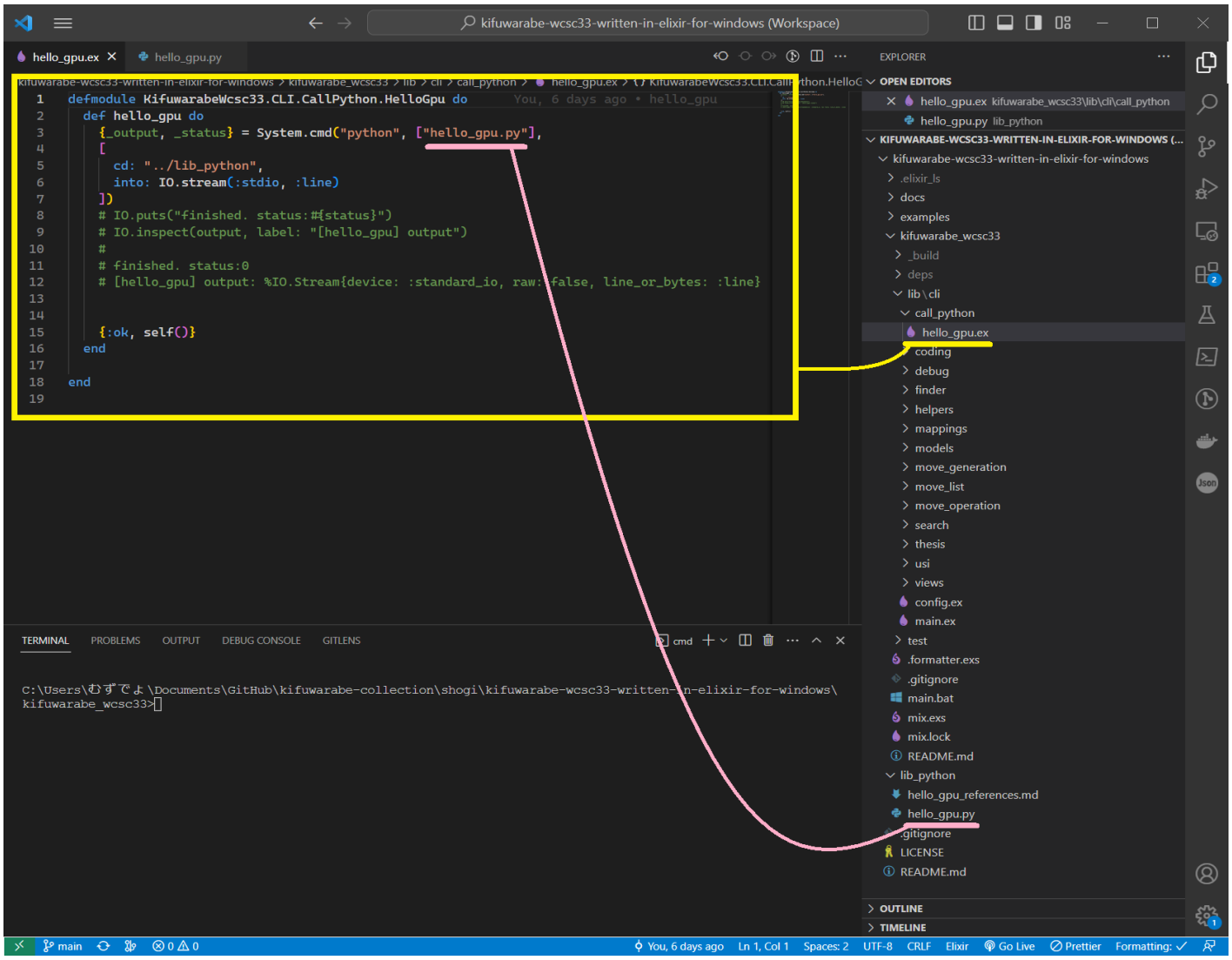
「 それでは　ヒャッホー　できないだろ。
エリクサー言語用のクーダ・ライブラリーは無いのかだぜ？」

「Ｗｉｎｄｏｗｓでコンパイルできるものが
見つからなかったぜ。
コンパイル・エラー」

「Ｌｉｎｕｘディストリビューションに
乗り換えないの？」

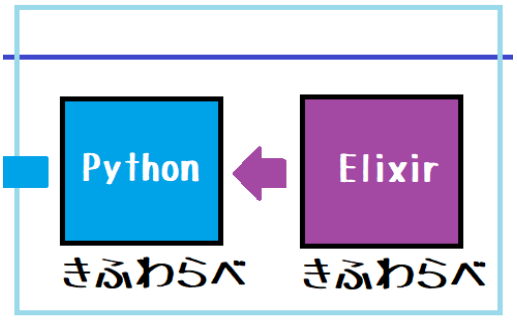
「Ｌｉｎｕｘディストリビューション用に　パソコンを　もう１台
用意するのも視野に入れ、　新しいモデルを買って　古いのをダブらせて
スレッドリッパーと　ジーフォースを　１セット　余らせようぜ？

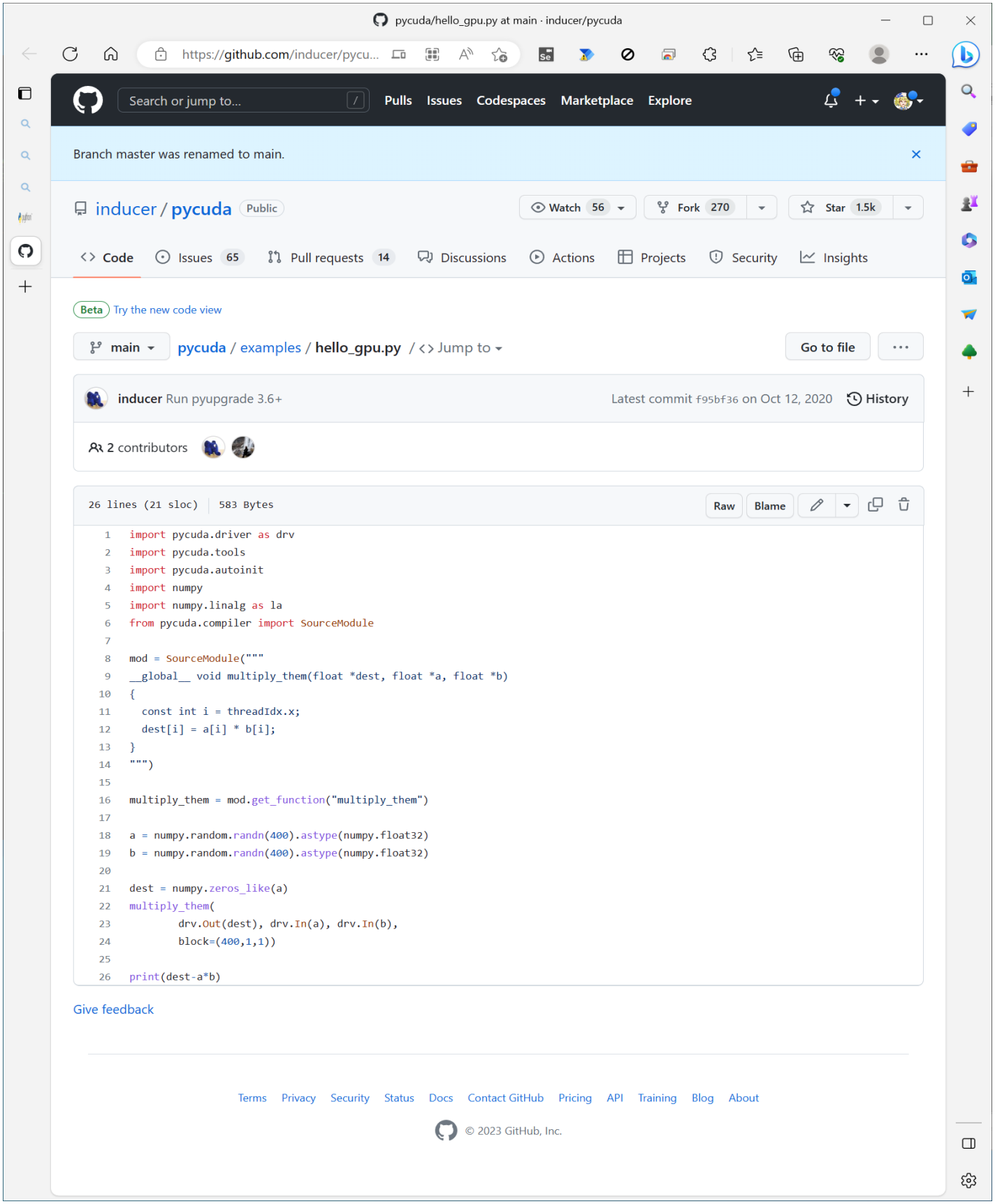
一度に　パソコンの部品を　買い揃えるのは　大変だしな」



「 上記が
エリクサーからパイソンを
呼び出している箇所だぜ。

hello_gpu.py というスクリプトを
実行している」



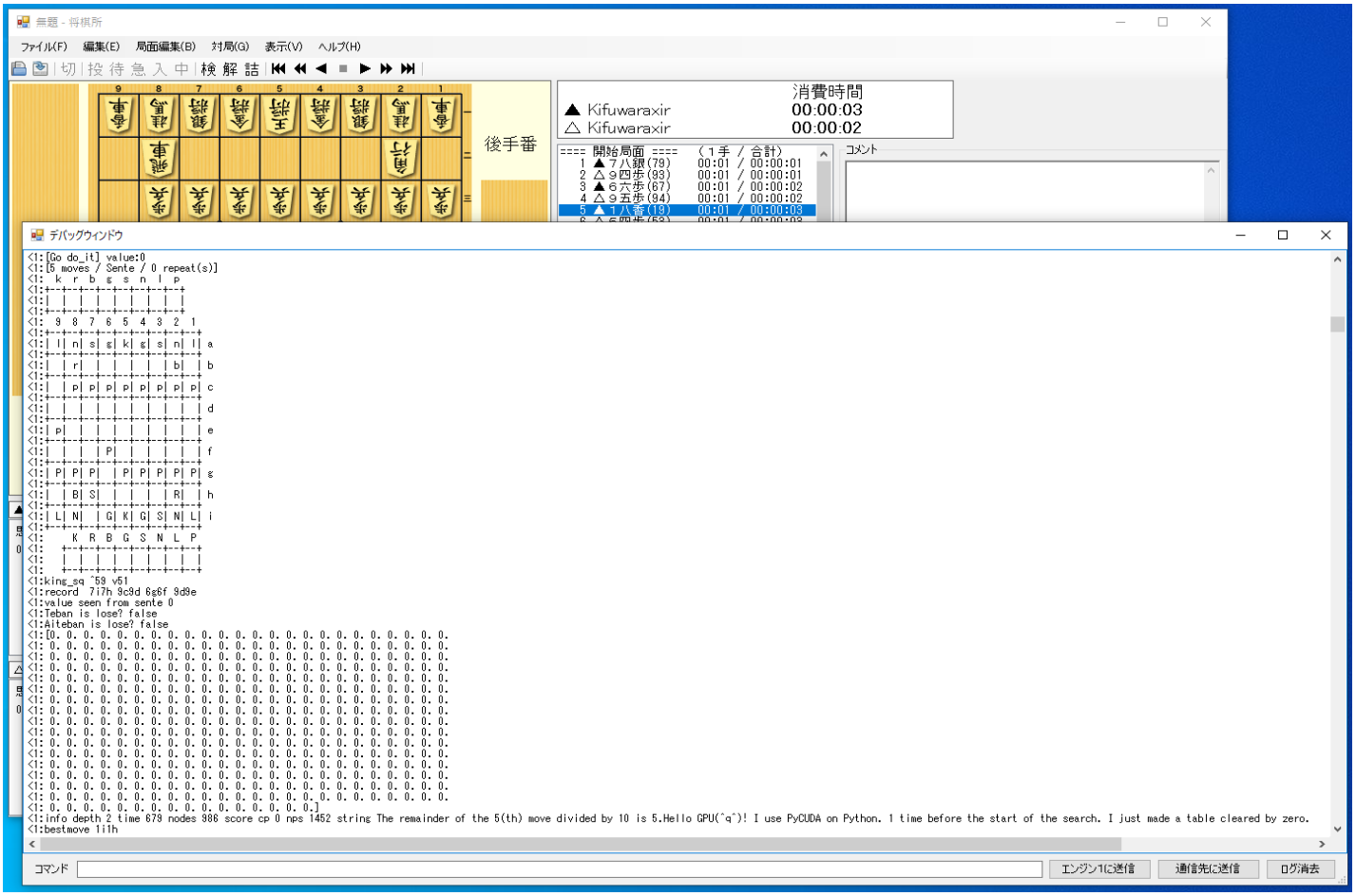


inducer/pycuda pycuda/examples/hello_gpu.py

https://github.com/inducer/pycuda/blob/main/examples/hello_gpu.py

「 hello_gpu.py は、 PyCUDA のエグザンプル・プログラムだぜ」

「 これを実行すると どうなんの？」



「 0 が 400 個 返ってくるぜ」

「 0 が 400 個 返ってくると どうなるんだぜ？」

「 満足だぜ」

「 大会中に　ハロー・ジーピーユー　を実行してたの？」

「 何手目かの数を１０で割った余りが　５か　６のときに
サーチ部に入る前に　１回　実行してるぜ」

「 ＧＰＵを使ったな」

「 使ったのかなあ？」

KANSUGATA GENGO

わたしは　関数型言語　を初めて使った

MITAINA

Elixir は 関数型「みたいな」言語

「今回のきふわらべは　エリクサー（Elixir）言語で実装したぜ。
ハロー・ジーピーユーを呼び出すとこだけ　パイソン（Python）言語を
使ったけど」

「エリクサーには　どんな利点があるの？」

「利点なんか　どうでもいいんだぜ。
関数型言語を使ったから　満足だぜ」

「ガチの関数型言語は　ハスケル（Haskell）が有名なんだが
Windows では環境設定すら難しかったので　ハスケルは止めた」

ながとダイアリー	Haskell将棋	Athlon 2650e	1.6GHz	1	1	1GB	WindowsXP	Haskell	Bonanza	リンク

第２１回世界コンピュータ将棋選手権

http://www2.computer-shogi.org/wcsc21/

「☝　ＷＣＳＣ２１　に　Haskell将棋（ながとダイアリー）　というのが
ハスケル言語で参加しているのを　大会帰りの電車で教えてもらったんで
きふわらべは　関数型言語で最初に参加したソフトではないぜ」

「ＣＳＡのＷｅｂサイトをぱっと見た感じ、
ＷＣＳＣの参加者で
エリクサーで実装された将棋エンジンは　わたしが初じゃないかな」

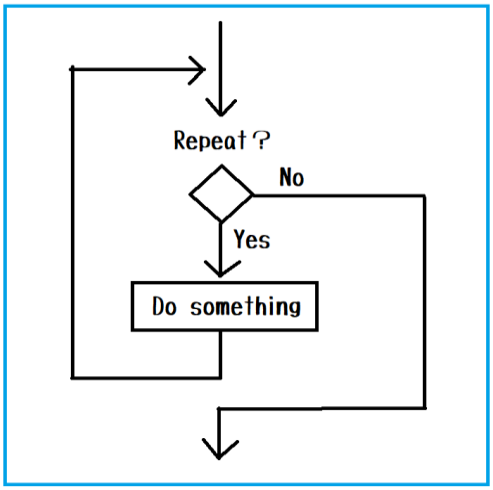
```elixir
defmodule KifuwarabeWcsc33.CLI.Search.Alpha do
  # 探索（深さへ再帰）
  def do_it(pos, depth, nodes_num_searched \\ 0) do
    move_list = KifuwarabeWcsc33.CLI.MoveGeneration.MakeList.do_it(pos)
    {move_list, pos} = KifuwarabeWcsc33.CLI.MoveList.ReduceSuicideMove.do_it(move_list, pos, :teban)
    {pos, best_move, value, nodes_num_searched} = choice_best(pos, move_list, nil, -32768, depth, nodes_num_searched)
    {pos, best_move, value, nodes_num_searched}
  end

  # 探索（水平へ再帰）
  defp choice_best(pos, move_list, sibling_best_move, sibling_best_value, depth, nodes_num_searched)

  defp choice_best(pos, [], sibling_best_move, sibling_best_value, _depth, nodes_num_searched) do
    {pos, sibling_best_move, sibling_best_value, nodes_num_searched}
  end

  defp choice_best(pos, [move | move_list], sibling_best_move, sibling_best_value, depth, nodes_num_searched) do
    pos = pos |> KifuwarabeWcsc33.CLI.MoveOperation.DoMove.do_it(move)
    nodes_num_searched = nodes_num_searched + 1

    {pos, opponent_value, nodes_num_searched} =
      if depth < 2 do
        opponent_value = lets_position_value(pos)
        {pos, opponent_value, nodes_num_searched}
      else
        {pos, _best_move, opponent_value, nodes_num_searched} = do_it(pos, depth - 1, nodes_num_searched)
        {pos, opponent_value, nodes_num_searched}
      end

    value = -opponent_value

    {sibling_best_move, sibling_best_value} =
      if is_alpha_update?(value, sibling_best_value) do
        is_uchifu_dume? = KifuwarabeWcsc33.CLI.Thesis.IsUchifuDume.is_uchifu_dume?(move, pos)

        if is_uchifu_dume? do
          {sibling_best_move, sibling_best_value}
        else
          {move, value}
        end
      else
        {sibling_best_move, sibling_best_value}
      end

    # 再帰（深さへ再帰）
    pos = pos |> KifuwarabeWcsc33.CLI.MoveOperation.UndoMove.do_it()

    if move_list |> length() < 1 do
      {pos, sibling_best_move, sibling_best_value, nodes_num_searched}
    else
      # 再帰（水平へ再帰）
      choice_best(pos, move_list, sibling_best_move, sibling_best_value, depth, nodes_num_searched)
    end
  end

  defp is_alpha_update?(value, sibling_best_value) do
    sibling_best_value < value
  end

  defp lets_position_value(pos) do
    pos.materials_value
  end
end
```

「 エリクサーで実装した　サーチ部だぜ。

インターネット上に　エリクサーで実装したコンピューター将棋の
エグザンプルが無いから　苦しんだ」

「エリクサーって
どんな言語なの？」

```c
// C
for ( int i = 0; i < 10; i++ )
{
    printf( "i=%d¥n", i );
}
```



Loop

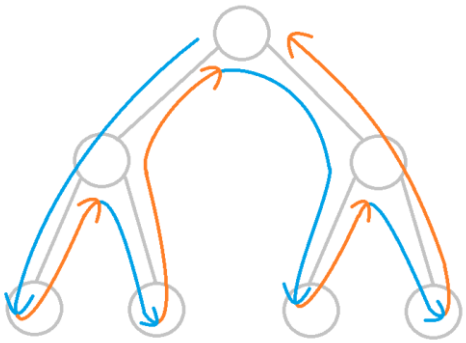
```c
// C
int i = 0;
while( i < 10 )
{
    printf( "i=%d¥n", i );
    i++;
}
```

```elixir
# Elixir
def do_repeat(i)

def do_repeat(i) when 10 <= i do
  nil
end

def do_repeat(i ¥¥ 0) when i < 10 do
  IO.puts("i=#{i}")
  do_repeat(i + 1)
end
```

「☝ フォー（for）ループも
ホワイル（while）ループもない。
ループは再帰関数で
書かないといけない」

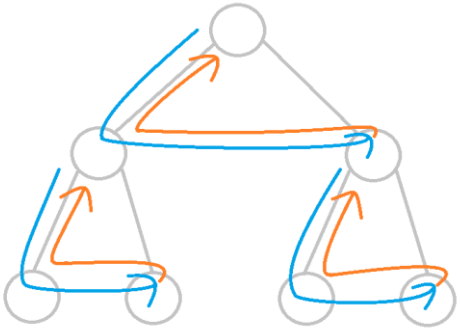
「ぜったい　関数でやってやる
という　やる気を感じるぜ」

「☞ 深さの方へ
再帰するのは
よくやるが……」

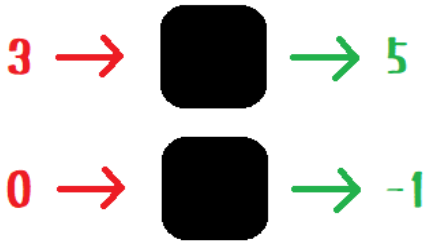


青い線：行き
オレンジ色の線：帰り

「☞ 水平にも再帰でやらな
ければいけないの
初めてだぜ」



「 なんで　むりやり　なんでもかんでも　関数でやるの？」



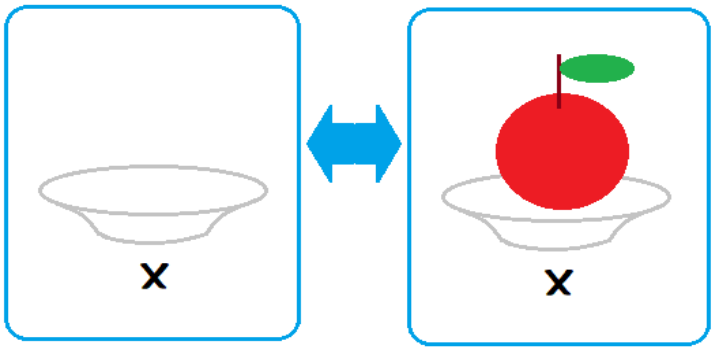
x × 2 − 1

Function

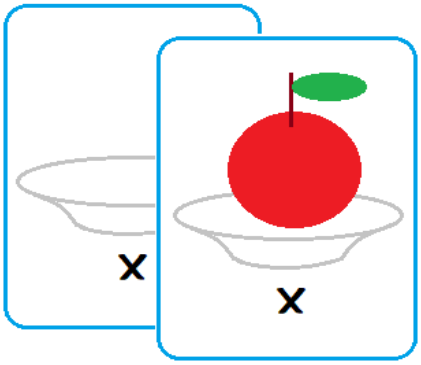
「 ☝　入力に対応する出力は　常に同じ、というのが
関数型言語の　特徴だぜ」

「 内部状態を持たない、とも言うぜ」

「 ローカル変数もないの？」



「 変数というのは　xという皿に　何もない状態、りんごがある状態と、
例えば　この場合　２つの状態を持つが……」

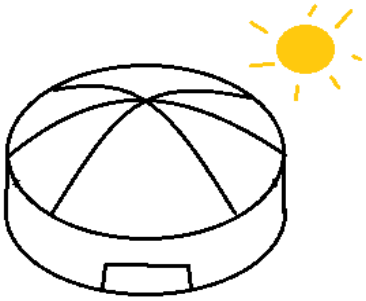


「エリクサーでは、　同じ名前で再定義する、　という理屈で
さっきの　x　と新しい　x　は名前が同じだけで　別物だから
xは　状態は持ってないんだ　と言い張ってるな」

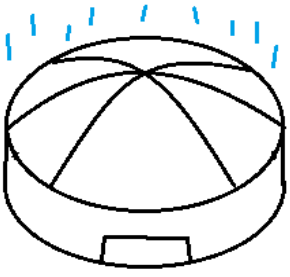
「それはそれで　メモリーの使用量を気にするなあ。
内部的に　どう　メモリーが使用されてるかまで　説明してくれなきゃ
コンピューター将棋のサーチ部の実装は　やってられないのよ」

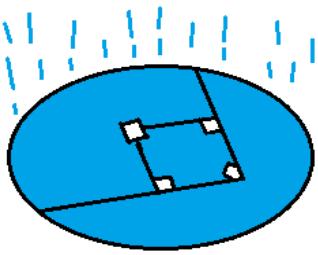
「なんでそんなに　状態を嫌うんだぜ？」

「☞　天気は状態だぜ。
雨が降ったら
野球は中止だぜ」



「ドームなら
天気という状態は
考えなくて済むだろ」





「考えることを
少なくしたいのか」

「不具合の原因が分からなくて　探すときなんかは
考えることが少ない方がいいわねえ」

HYAHHO-

プログラミングで　ヒャッホー　しようぜ？

SYURUI
６種類の　プログラミング・パラダイム

「 わたしが　いつの頃だったか、
海外の伝説的なプログラマーが、誰だったか忘れたが、言ったことが
記事になったものを読んで　頭に残ってるんだぜ」

プログラマーなら　少なくとも
６種類のプログラミング言語を　やるといい

プロシージャ型言語
オブジェクト指向型言語
関数型言語
論理型言語

「 あとは何だったか　忘れた」

「 覚えてろよ！」

「 プログラミング・パラダイム（Programming paradigm）で
検索すると、
６個どころではなく　沢山出てくるんだけど」

「 色々あるという　ヒント　が示されただけで　充分だぜ」

「 今大会のわたしは
ＣＰＵの　パラレル・プロセッシング　を使うのと、
ＧＰＵの　パラレル・プロセッシング　を使うのと、
プログラミング・パラダイムの　関数型言語　を使うという
３つの使うの　ヒャッホー　が達成できた実りの多い大会だったな」

「 達成なのかなあ？」

Git Hub　きふわらべＷＣＳＣ３３
https://github.com/muzudho/kifuwarabe-wcsc33-written-in-elixir-for-windows

「 ☝ この Git Hub の 266コミット目が　今大会に出た　きふわらべ　だぜ。
コードは　MITライセンス　にしてある。好きに使えだぜ。
わたしの中での識別名は　きふわらくさ（Kifuwaraxir）　だぜ」

「 草」

```
>>> Dad, I want you to use
                the computer
you bought to the fullest
                (Grass)
```