Stackの応用
関数のスコープ &
逆ポーランド記法

情報科　飯島　涼

データ構造

プログラムの中で，データをどのようにして格納したり，取り出したりするかという構造のこと

代表的なデータ構造

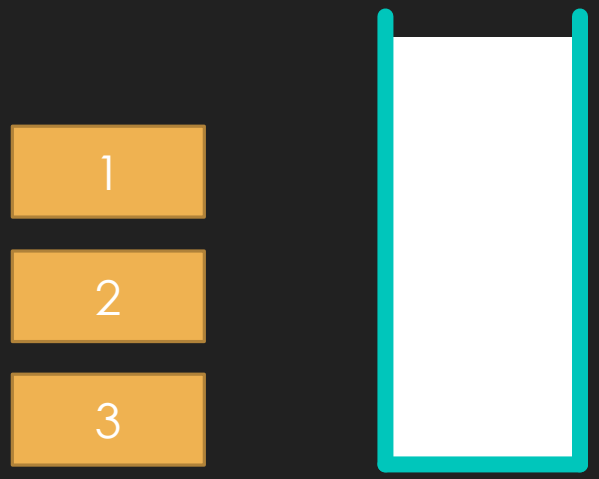
- Stack
- Queue
- リスト
- 辞書

など

スタック・キューとは

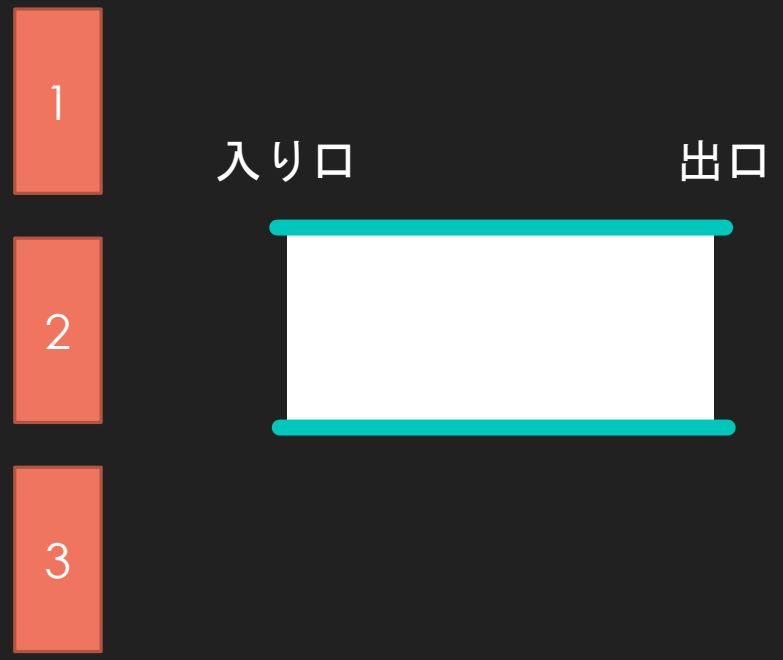
スタック (Stack, Last In First Out, LIFO)

- 1箇所からのみ出し入れができる箱



キュー (Queue, First In First Out, FIFO)

- 入り口，出口がそれぞれ別に1つずつ固定された箱



スタック・キューを何に使うのか?

スタック (Stack)

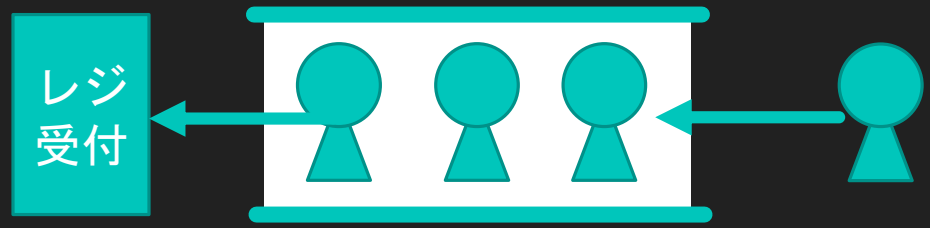
- メモリやレジスタの容量を節約したい計算
  - 逆ポーランド記法による計算
  - 関数のスコープ（Stackを用いると，メモリを節約しながら簡単に実装できる）

キュー (Queue)

- 待ち行列の解析（待ち行列理論）
  - リアルな待ち行列
  - 交通渋滞
- オンラインリアルタイム動画(Zoom，Skypeなど)のパケット受信

Queueの利用用途

- 待ち行列の解析（待ち行列理論）
  - リアルな待ち行列
  - 交通渋滞



- オンラインリアルタイム動画(Zoom，Skypeなど)のパケット受信

同じ現象として解析が可能になる

シミュレーション実験

- 実際に交通渋滞や，客の到着を予測するのに使用されるシミュレーションをやってみる

  - 関連分野:
    - 数学（統計）
    - 心理学
    - 待ち行列理論
    - ネットワーク解析
    - 交通渋滞

客の到着シミュレーション

- シミュレーションの仮定
  - 流行りの廃れたタピオカ屋
  - 1秒あたりの客の到着確率を，p=0.05とする.
  - 客は，ランダムに1人ずつ到着する (一緒の来店は1人とカウント)

- プログラム上の規定
  - random 関数を用いて、客がランダムに到着する様子をシミュレーションする
  - [1秒経過すること=for文の1回分のループ] と置き換える

- 導きたいもの
  - 最後の客が到着してから、次の客が到着するまでにどのくらいの時間が経過するか? (=客の到着時間間隔)
  - 客の到着時間間隔 < サービスを提供する時間　となると行列が発生し、レジの増員・効率化・機械化が求められる

【授業内演習】客の到着シミュレーション

以下の要求にしたがってプログラムを書いてみてください。

新しくタピオカ屋を開店するために、原宿におけるタピオカ屋の客の到着をシミュレーションすることにしたRは、以下の要件で客の到着をシミュレーションすることにした。

- 1秒あたりの客の到着確率は，p=0.05であることがわかっている.
- 客は，ランダムに1人ずつ到着する (一緒の来店は1人とカウント、整数時間ぴったりにしか来ないと考えてよい)
- for文が1回まわるプログラムの動作を、1秒の経過とみなして、TIME_END=60秒シミュレーションする。
- 客の到着間隔(最後の客が到着してから、次の客が到着するまでの時間)を変数など(time_dif)で管理し、リストに格納していく(time_dif_list など)。
- time_dif_listの要素をヒストグラム化する。
- ヒストグラムができたら、TIME_END秒は、60, 360, 1000, 3600 など、だんだん増やしながらヒストグラムの形の変化を観察してください。

方針

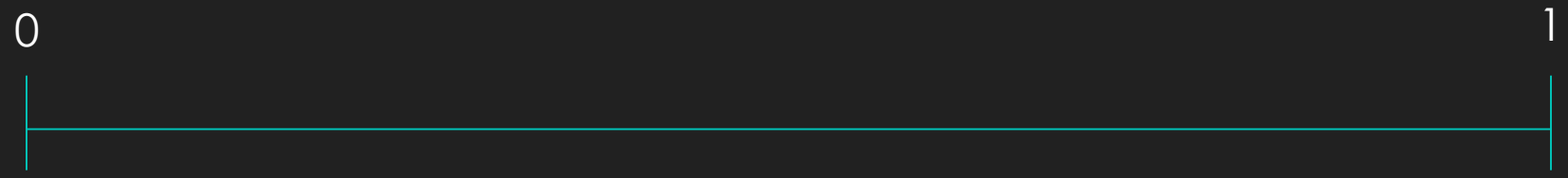
- 欲しい情報は全て保存する
- 今まで習った保存の形式
  - 変数
  - リスト

方針

- 欲しい情報
  - 客が到着してから，次の客が到着するまでの時間
- 保存のやり方　アイディア
  - 変数を使う場合
    - 次の客が到着したと判定されるまで，time_difに+1し続ける
  - リスト
    - 客の到着時間を格納するリストを別に作っておく

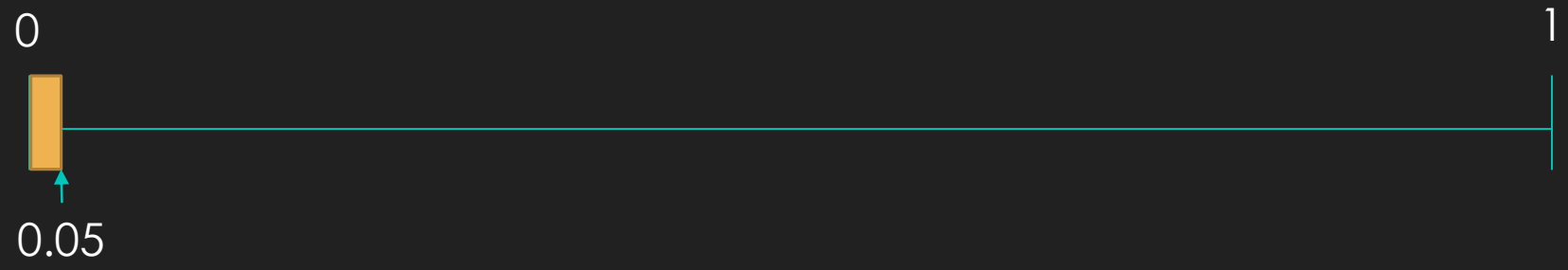
客の到着の判定

random関数を利用



0-1の数直線のうち，どこか1箇所を選ぶ関数とみなすこともできる

[0-1のうち，0.05以下が選ばれたら] = 5%の確率で



本日の内容

- Stackの応用（教科書記載なし）
  - 関数のスコープ
    （世界標準MIT教科書 Python言語によるプログラミングイントロダクション第2版: データサイエンスとアプリケーション, p.43 – p.46 / 入門 Python 3,　p.128 - 131）
    - スコープとStack
  - 逆ポーランド記法
    （プログラミングコンテスト攻略のためのアルゴリズムとデータ構造, p.82-86）
    - 逆ポーランド記法とStack

関数のスコープ

- スコープ（名前空間）

  変数が有効な部分のこと

  原則は3つ

  1. 局所変数(local variable): 関数の中で定義した変数は，関数の外では使えない

  2. グローバル変数(global variable): どの関数にも属していない変数は，どこでも使える

  3. 局所変数優先の法則(local variable priority) :
     同じ名前の局所変数と，グローバル変数がある場合は，局所変数が優先される

なぜスコープを学ぶのか？

- スコープの勘違いが，バグの原因の上位に必ず入るから
  - 仕事にしているプロでも見落とすことがあるバグの一種になる（local variable & global variable）
  - スコープの考え方を理解して，あらかじめバグの原因を減らすコードの書き方を知るため

グローバル変数(Global variable)

- どの関数の中にも含まれない変数のこと
- 性質: すべての場所で参照することができる（同じ名前の局所変数がある場合は条件あり、後述）

例（なぜエラーにならないのか?）

```
# 関数の中に書いた変数は、定義より手前にあるのでエラーになりそうだ
def f(x):
  print(a, b)
  return 2*x

a = 2
b = 1+2*5
c = f(5)
```

```
2 11
```

局所変数 (local variable)

- 関数の中に定義された変数のこと
- 性質: 関数の中で定義された変数を，関数の外で使うことはできない

例 (バグ/エラーの原因その1)

```
1 # returnを使わずに、yに代入して受け取ろうとした
2 def f(x):
3     y = x * 2
4
5 f(10)
6
7 print(y)
```

関数の中

関数の外

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ba4d63dc8d28> in <module>()
      5 f(10)
      6
----> 7 print(y)

NameError: name 'y' is not defined
```

関数の外からは
定義されていない扱いとなる

局所変数とグローバル変数の衝突

- 同じ変数名の局所変数/グローバル変数があったらどうなるか? (バグ/エラーの原因その2)

【授業内演習】どのように出力されるだろうか? p.13の原則を見返しながら、なぜそのように出力されたか説明できるか?

```
1 def f(x):
2   y = 1
3   x = x + y
4   print("x=", x, "関数内")
5   return x
6
7 x = 3
8 y = 2
9 z = f(x)
10 print("z=", z)
11 print("x=", x, "関数外")
12 print("y=", y)
```

●再生　ここまで
次からは興味のある人向け
自由課題をやりたい人は、
p.25まで飛ばしてください

【応用】スタックフレームとは

- 変数名，関数名と値の対応を保存しておく表（Table）のこと
- スタックフレームが作られるタイミング
  - プログラムの実行開始時
  - 関数の呼び出し時

スタックフレームの説明

① グローバル変数用のスタックフレームができる

```
1 def f(x):
2    y = 1
3    x = x + y
4    print("x=", x, "関数内")
5    return x
6
7 x = 3
8 y = 2
9 z = f(x)
10 print("z=", z)
11 print("x=", x, "関数外")
12 print("y=", y)
```

Stack

名前	値
x	3
y	2
z	

スタックフレームの説明

②関数が呼び出されると、あたらしくスタックフレームが作られる (push, put)

```
1 def f(x):
2   y = 1
3   x = x + y
4   print("x=", x, "関数内")
5   return x
6
7 x = 3
8 y = 2
9 z = f(x)
10 print("z=", z)
11 print("x=", x, "関数外")
12 print("y=", y)
```

Stack

名前	値
x	3
y	1

名前	値
x	3
y	2
z	

スタックフレームの説明

③ 変数の値変更や呼び出しがあるたびに、

Stackの一番上にあるスタックフレームを参照して値を受け取る

```
1 def f(x):
2   y = 1
3   x = x + y
4   print("x=", x, "関数内")
5   return x
6
7 x = 3
8 y = 2
9 z = f(x)
10 print("z=", z)
11 print("x=", x, "関数外")
12 print("y=", y)
```

Stack

名前	値
x	3 => 3+1=4
y	1

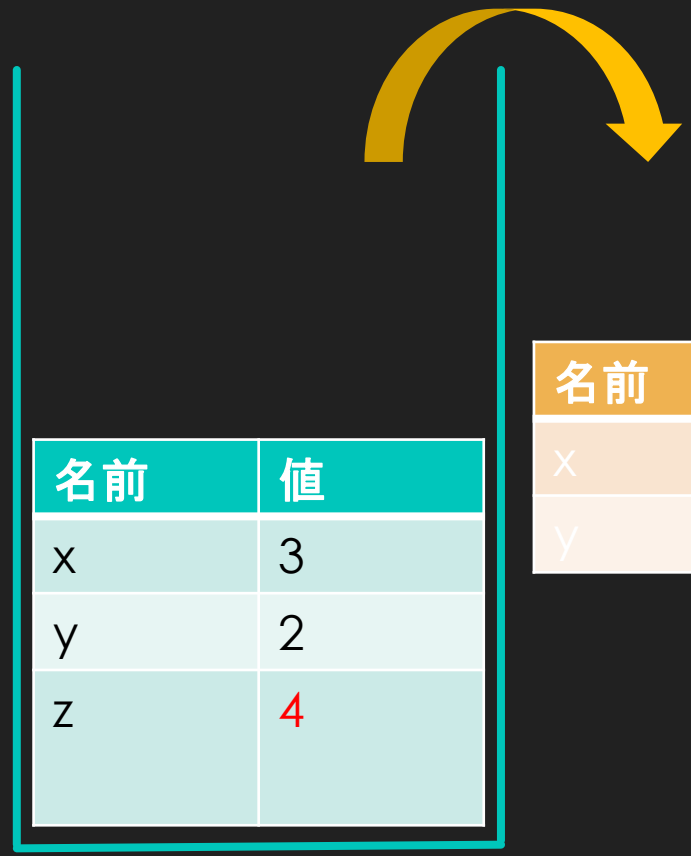
名前	値
x	3
y	2
z	

スタックフレームの説明

Stack

③ 関数が終了すると、その関数開始時に作られたスタックフレームは解放される(pop, release, get)

```
1 def f(x):
2   y = 1
3   x = x + y
4   print("x=", x, "関数内")
5   return x
6
7 x = 3
8 y = 2
9 z = f(x)
10 print("z=", z)
11 print("x=", x, "関数外")
12 print("y=", y)
```



名前	値
x	3
y	2
z	4

名前	値
x	4
y	1

スタックフレームの説明

④同様に、Stackの一番上のスタックフレームを参照して、出力する

```
1 def f(x):
2    y = 1
3    x = x + y
4    print("x=", x, "関数内")
5    return x
6
7 x = 3
8 y = 2
9 z = f(x)
10 print("z=", z)
11 print("x=", x, "関数外")
12 print("y=", y)
```

Stack

名前	値
x	3
y	2
z	4

スタックフレームを使う意義

余計にコンピュータのリソース(メモリ)などを使わないようにするため

コンピュータが重くなっている（熱くなっている）ときの
イメージ　→

原則:
使わなくなったデータは解放する (スタックフレームの解放)

バグを減らす工夫

- 局所変数と、グローバル変数に、意味もなく同じ変数名を付けない
  - どっちの変数が呼び出されているのかわからず、エラーにはならないのに意図しない挙動をすることがある
- 変数名には、何を保存しているのかわかりやすい名前を付ける
  - x, y, zや a, bなどは、実用的なアプリでは推奨されない（授業ではわかりやすさのためにそうしているだけ）

【応用】逆ポーランド記法

- 式の()（かっこ）による計算をなくして、計算の順番を自動で判定しやすくする書き方
- かっこの判定を実装するのは難しい（括弧の中の括弧の中の… …）
- Stackと組み合わせて使うことで、どの数とどの数に演算を適用すればいいかの判断を自動でやってくれるようになる。

数式の記法について

中置記法

- 普段私たちが使っている式
  - 3 + 4
  - (3 + 4) * (2 - 7)
- 括弧の中身の数を先に計算して置き換える。
  - 先に扱われる演算子ほど、括弧の内側に入っている。
  - ⇒ 括弧の対応の判定が面倒

逆ポーランド記法 (後置記法)

- プログラムにとって計算しやすい式
  - 3 4 +
  - 3 4 + 2 7 - *
- 演算子が現れたら、手前の2つの数字をその演算子で計算して置き換える。
  - あとに扱われる演算子ほど右に来るように書く
  - ⇒ Stackを利用すると、計算の順序を気にせずに計算する数を判断できる

自由課題 逆ポーランド記法用の電卓を作る

- Aizu Online Judge ALDS1_3_A Stack
  - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_3_A&lang=ja　より問題文引用
  - 逆ポーランド記法は、演算子をオペランドの後に記述する数式やプログラムを記述する記法です。例えば、一般的な中間記法で記述された数式 (1+2)*(5+4) は、逆ポーランド記法では 1 2 + 5 4 + * と記述されます。逆ポーランド記法では、中間記法で必要とした括弧が不要である、というメリットがあります。
  - 逆ポーランド記法で与えられた数式の計算結果を出力してください。

AtCoderの類題

Stackがなくても解けるが、わかっていると考えが早く思いつくもの

- https://atcoder.jp/contests/abc120/tasks/abc120_c
- https://atcoder.jp/contests/abc043/tasks/abc043_b