Algorithms (2022 Summer)
#1：イントロダクション，計算量

矢谷 浩司

ご登録をお願いします！

https://forms.gle/M8kDy5sYxXwEMAeb8
今すぐお願いします！

講義のslackやコードチャレンジで使うシステムへの招待を行う他，皆さんのバックグラウンドを把握するため，ご協力をお願いします．

単位の取得が必要な人は，別途UTASでの登録を忘れないようにお願いします．

Welcome!

この講義を担当する矢谷浩司です.

今学期どうぞよろしくお願いいたします！

アルゴリズムとは？

アルゴリズムとは

あるタスクを達成するために設計された有限回の計算手順（ソート，サーチ，最適化などなど）．

アルゴリズムは正しい，あるいは「最適な」解を導くように設計されている．（ただし，最適と言っても，あくまでアルゴリズム内で設定された評価基準による）．

多くの場合は（時間コスト，メモリコスト，通信コストにおいて）効率的な計算手順を意味する．

アルゴリズムの例

文章の中から所望の単語がある場所を探し出す.

たくさんの数字を小さいもの順に並べる.

A駅からB駅に行くまでの電車でのルートを検索し,
運賃の安い順に並べる.

進学選択で各学生さんの配属学科を決める.

アルゴリズムがわかると

効率的な処理を設計できる.

与えられた処理がどの程度計算を要するものなのかを見積もることができる.

ボトルネックになっているコードを解析してそこを改善できる.

この講義でやること

データ構造やアルゴリズムに関する基礎知識の学習

それらをpythonで実装し，実際に体験

さらにそれらを利用して，演習課題に取り組む

この講義で学んでほしいこと:

頭で考えた処理手順をコードに落とし込む

この講義を終えた後にはこうなってほしい

データ構造やアルゴリズムに関する基礎的な話がわかっている.

それらをpythonで実装することができる.

それらを応用するような課題に自分で取り組める.
（例えば，競技プログラミングなど）

この講義のモットー:

Let's code! 😆

講義内容を設計するにあたって

アルゴリズムを初めて学ぶ人が大部分，という前提です.

コーディングを得意としない人にも理解してもらえるように，かなり噛み砕いてスライドを作ったつもりです.

また，Let's code!のモットーの下，毎週手を動かす機会を提供するようにしています.

この講義の対象者

アルゴリズムを初めて学ぶ（もしくはそれに近い）．
　　勉強してみたが，挫折した人もvery welcome!

プログラミング，Pythonの基本は一応理解している．

競技プログラミングとか，ちょっと始めてみたいかも，と思っている．

この講義の対象者でない方😅

「蟻本読破しました．」
「暖色コーダーです．」
「プログラミングコンテスト入賞者です．」

上記方々はこの講義を受けても多分非常につまらんと
思いますので，ぜひ我が道を行ってください！

この講義が（積極的に）カバーしないこと

アルゴリズムの正当性の証明

計算量の細かい議論，証明

数学的理論

この講義は，アルゴリズム・データ構造を直感的に理解して，手を動かすことを重視するスタイルですので，理論や説明の厳密性には目を瞑ってあげてください.🤦
（説明は丁寧に行うよう心がけています．）

講義体制

講義担当者：矢谷 浩司
TA：加賀谷 昂輝，廣澤 佑亮，利根 悠司

ホームページ：https://iis-lab.org/algorithms
メールアドレス：algorithms@iis-lab.org（矢谷＆TA）

基本は講義のslackにてコミュニケーションをお願いします．

受講に必要なデバイス，サービス

PC

OSは問わない

ブラウザとしてChromeかFirefoxを推奨

インターネット環境（オンラインで受講する場合）

ECCS/utacアカウント

講義の前提条件

使用言語：Python3
コーディング環境：track（後で説明します）

学科PCか自前のPCを持参し，ネットワークに接続できる状態にしておいてください．

ローカル環境でもpythonを実行できるようにしておいてもらえるとよいです（必須ではありません）．

講義の前提条件

プログラミング，pythonの基礎的内容は身についているものとします.

変数の型，配列，条件分岐，ループ，関数，
再帰，入出力などなど.

このあたりが不安な人は今のうちに自習をお願いします.

講義

スライドを使って説明します．スライドは授業のホームページにもアップロードしておきます．

授業開始時刻5分前くらいから，Zoom meetingを開始します．マイクはONにできないよう設定されています．

終了後も後から視聴できるようにしておきますので，見逃した方はお時間のある時に自習してください．

講義

246教室にて受講していただくことも可能です.

246教室で受講を希望する方は，モバイルアプリMOCHAをご自身のスマートフォンにインストールし，事前に入室予約を行ってください.

事前に入室予約を行っていない方には，教室の混雑状況によってその場で退室をお願いすることがあります.

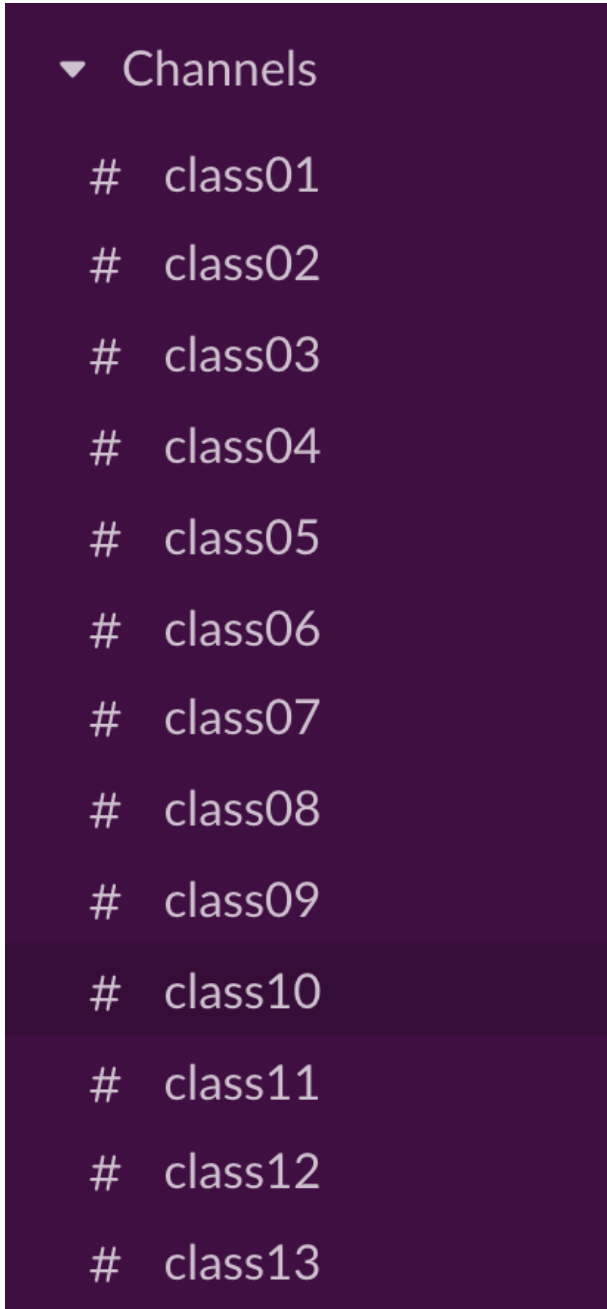
Q&A

講義のslack（EEICのslackとは別）に皆さんを招待しています.

各講義回チャネルを用意していますので,
そちらでコメントをお願いします.

/anonymousと打つと，無記名で投稿できます.

教室に来ている方もslackでご質問ください.



Slackのテストを兼ねて

今みなさんが特にハマっていること（食べ物，飲み物，アクティビティ，趣味，サークルなんでも）を教えてください！

#class01に投稿をお願いします．/anonymousをつかって投稿してもらってもOKです．

成績評価

コードチャレンジ基本課題（全24課題）：34点

選択課題（どちらかを選択，両方やった場合はより得点の高い方を成績に利用）：33点

- コードチャレンジExtra課題（全11課題）
- レポート課題（全2課題）

期末試験：33点

成績評価

ただし，以下の制約に1つでも引っかかる場合，この講義の成績は「未受験」という扱いになります.

- コードチャレンジ基本課題のうち，未着手のままの提出の回数が4回以上.
- Extra基本課題，レポート課題のどちらとも全て未提出.
- 期末試験を放棄（出席しなかった）.

コードチャレンジ

2種類あります.

基本課題

Extra課題

コーディングする時間を定期的に作っていただくため,
毎週お届けします.

課題の配信は終了15分くらい前, 締切は直近の月曜日の21:00です.

基本課題：全員できてほしい課題

必修課題ですので，全員提出してください．

主に，講義内で紹介された重要なアルゴリズムの実装や変更を行うものです．

授業の内容を追えばできるようになっています．

平均的には1〜1.5時間程度で完成させられるくらいのボリューム感．

Extra課題：頑張ればできる腕試し的課題

選択課題の1つ.

授業の内容を踏まえた発展的な課題で，平均的には
6時間程度で完成させられるボリューム感.

ちょっと競技プログラミング的なノリを含んでいます.

コーディング能力をさらに高めたい人はぜひチャレンジ
してみてください.

コードチャレンジ

締切は直近の月曜日の21:00ですが，「土日も課題に取り組め」というメッセージではございません．😅

お時間のある時を自由に選んで，できる限り課題に
取り組むことのできる環境を提供するための設定
ですので，ご理解ください．

土日に取り組むことを妨げることはありませんが，
しっかりリフレッシュする時間を取ることも意識し，
自分の生活・学習リズムを作ってもらえると嬉しいです．

レポート課題

もう1つの選択課題.

既に掲載していますので，ご参照ください.

コーディングに自信がない人でも，内容をしっかり理解して取り組めば，良い点に繋がります.

Extra課題とレポート課題の採点

どちらが特別に不利にならないように平均点，点数分布がある程度近くなるように，レポート課題の採点は調整します．

ただし，本質的に全く違う課題であり，提出数も大きく違うことが予想されますので，全く同じ得点分布になることは難しいとご承知おきください．

期末試験

現時点では，講義室にて実施する予定です．

70〜80分程度の試験を予定しています．

ただし，今後の状況によって変更することもあり得ますので，授業中，slackでのアナウンスに従ってください．

講義内容

#1: イントロダクション，計算量
#2: 累積和，整数関連
#3: データ構造
#4: 探索（サーチ）
#5: 整列（ソート）
#6: 文字列照合
#7: 動的計画法1
#8: 動的計画法2

講義内容

#9: BFS，DFS

#10: グラフアルゴリズム1（最短経路問題）

#11: グラフアルゴリズム2（最小全域木，トポロジカルソート）

#12: グラフアルゴリズム3（最大流問題，最小費用流問題，二部グラフのマッチング問題）

#13: 「難しい問題」とは，さいごに

（特別講演があるかも？）

出席

取りません. 😆

授業に出席したり，講義の配信や録画を視聴したりすることは必須ではありませんが，頑張ってお届けしますので，見ていただけると嬉しいです.

励ましのお便りもお待ちしております！

出席せずとも，課題には積極的に取り組んでください.

病欠，公欠

以下の情報を**メールにて**，algorithms@iis-lab.orgに送ってください．認められた場合，課題提出期限をこちらが指定する日時まで延長します．

- 病欠，公欠を希望する授業日
- 病欠，公欠の理由
- 理由が正当なものであることを裏付けるもの（診断書，処方箋，出席する学会のプログラム等）

可能な限り事前に，事後でも速やかにお願いします．大きな怪我や病気などの例外を除き，病欠，公欠を希望する授業日から1週間以上を経過した場合は，認めないものとします．

TAさん

講義の配信が見れない，trackが使えないなどのトラブルがあればslackで連絡ください．受講者が多い中で対応をお願いしていますので，その点ご理解ください．

みなさんのコードに対する個別のデバッグ等はサポートできかねますので，ご了承ください．

Academic misconduct

この講義でも厳しく対処します.

剽窃，無断流用などが発覚した場合，その度合いに
応じて以下のような処置が取られます.

- 提出された課題，試験を0点とする.
- 提出された課題，試験を0点とし，以降の提出を認めない.
- 全課題，試験に遡って0点とし，以降の提出を認めない.

Academic misconduct

コードチャレンジでは皆さんに課題に対するコードを提出してもらいます.

Extra課題に関しては，提出されたコードに対して後日類似度チェック等を行い，明らかに類似したコードがあった場合は，その全てに対してペナルティを課します.

基本課題に関してはほぼ同じようなコードになるはずですので，このルールを直接は適用しませんが，チェックは行います.

Academic misconduct

講義で使うシステムではコードをテストケースにかけた時点で，自動でバージョン管理します．

コピペ等を行なった場合には，不自然にコード行数が増えたり，テストケースにパスする回数が突然増えたりするので，その辺りもチェックします．

Academic misconduct

本やWeb上の情報を参考にするのは構いません．ただし，そのまま使うことのないようにしてください．

友達同士で教え合うのも積極的にどうぞ．ただし，コードを直接見せ合うのではなく，考え方だけを共有するようにしてください．

「自分で手を動かす」ことをご自身で大切にしてください．

事前にお伝えしておきたいこと

6月に第2子が生まれる予定です．👶

全13コマの内，いくつかは録画を配信する形になる可能性があることをご承知おきください．（コードチャレンジは毎週配信する予定です．）

ただし「対面・オンライン併用型A（総時間数の半数以上を対面で実施）」の制約は十分に満たす予定です．

つまり，文科省的には「対面授業」となる予定です．

ご意見ください

何か気がついたこと，改善できたらいいのに，と思ったことなどあれば，ぜひ教えてください．

随時皆さんに授業に関するアンケートをお願いします．
ぜひご協力ください．🤦

1回目はゴールデンウィーク前後を予定しています．

登録フォームにあった質問・コメント

「学部横断型プログラム修了を目的として履修を検討していますが、他学部履修は問題ないでしょうか。」

→こちらとしては問題ありませんが，この講義の単位がプログラム修了に考慮されるかどうかは，ご自身で確認をしてください．

「専修の必修科目・実習が不定期で入るため、録画視聴となる回が一定数見込まれるのですが、特に問題ないでしょうか。」

→問題ありません．

登録フォームにあった質問・コメント

「先ほどと同様の理由から、期末試験を受験できない可能性があります。後日日程が決まり次第、受験に支障が出る様であれば改めてご連絡します。」

→単位のためには期末試験を受験することは必須となります．期末試験の日程は電子情報工学科・電気電子工学科の日程で調整しますので，他学部・他学科の方はそれに合わせていただくようにお願いいたします．日程が合わないという理由での救済措置は予定しておりません．

登録フォームにあった質問・コメント

「ぜひ頑張りたいのですが、ソフトウェア２の課題に非常に苦しんだ自分についていけるのか少し不安です」

「競プロ経験者になんとか食らいついていければなと思います。よろしくお願いします。」

→マイペースにやりましょう！わかったり，できたりする手応えを可能な限り味わってください．コーディングが難しい場合でもレポート課題でご自身のオリジナリティを発揮してもらえればと思います．

では，早速！

アルゴリズムとは（再掲）

あるタスクを達成するために設計された有限回の計算手順（ソート，サーチ，最適化などなど）．

アルゴリズムは正しい，あるいは「最適な」解を導くように設計されている．（ただし，最適と言っても，あくまでアルゴリズム内で設定された評価基準による）．

多くの場合は（時間コスト，メモリコスト，通信コストにおいて）効率的な計算手順を意味する．

アルゴリズムの最低条件

有限実行時間で必ず止まる（停止性）.

止まった時，正しい結果が得られる.

アルゴリズムの最低条件

有限実行時間で必ず止まる（停止性）.

止まった時，正しい結果が得られる.

まあ，そりゃそうじゃないと困りますよね. 😅

気にしたいのは「いつ止まる」のか.

計算量

与えられたアルゴリズムがどの程度の時間的・メモリ的コストで実行できるかの目安.

具体的な時間やメモリ量を表すものではなく，必要なコストを半定量的に表す指標.

大雑把な見積もり，という感じ.

計算量

与えられたアルゴリズムがどの程度の時間的・メモリ的コストで実行できるかの目安.

具体的な時間やメモリ量を表すものではなく，必要なコストを半定量的に表す指標.

$O(n)$ とか $O(n^2)$ という形で表す.（「オーダーn」というふうに読む.）ビックオー記法と呼ばれる.

時間計算量の例（線形探索）

ある配列の中から，別に与えられた値に一致する要素を探し出す．

配列の先頭から順にチェックして，一致する要素があればその時のindexを返す．ない場合は-1を返す．

例）
入力：[8, 3, 4, 1, 6, 9, 2]で6の場所を探す
出力：4

時間計算量の例（線形探索）

```
def search(sequence, key):
    i = 0
    while i < len(sequence):
        if sequence[i] == key:
            return i
        i += 1
    return -1
```

時間計算量の例（線形探索）

```
def search(sequence, key):
    i = 0
    while i < len(sequence):
        if sequence[i] == key:
            return i
        i += 1
    return -1
```

平均的な実行回数

1回
n/2回
n/2回
1回
n/2回
(1回)

時間計算量の例（線形探索）

```
def search(sequence, key):
    i = 0
    while i < len(sequence):
        if sequence[i] == key:
            return i
        i += 1
    return -1
```

$O(n)$

一番支配的な項のみで計算量を表す．

計算量

入力が$O(n)$規模である想定（n個の要素の配列とか）.

平均的な場合と最悪の場合で計算量が変わることもある.

両方の場合で分けてたり，最悪の場合だけ考えたり，
とその時々で違う.

表されている計算量がどんなケースを考えているもの
なのか，正しく理解してから前に進んでください.

オーダーの感覚

$O(1)$
$O(\log n)$
$O(n)$
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$
$O(\log n)$
$O(n)$
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$ 😩
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$ 😩
$O(2^n)$ 😱
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$ 😩
$O(2^n)$ 😱
$O(n!)$ 😇

(データの大きさによります)

オーダーの感覚

$$O(n^2)$$

$$O(n \log n)$$

$$O(n)$$

オーダーの感覚

$O(n^2)$

$n = 10^6$

$O(n \log n)$

10^8ステップ/秒

$O(n)$

オーダーの感覚

$O(n^2)$ 2.7時間

$n = 10^6$

$O(n \log n)$ 200ミリ秒

10^8ステップ/秒

$O(n)$ 10ミリ秒

ビックオー表記

$T(n) = O(f(n))$という形で表す.

十分大きなnに対して，関数$T(n)$が，$f(n)$に比例，もしくはそれより小さいことを表す.

漸近的上界，ともいう.

この漸近的上界はいくらでも存在するが，できる限り精度良く＆簡潔に表現しているものを通常は採用する.

ビックオー表記

例えば，$T(n) = n^2 + 4n + 100$の場合，$O(n^2)$，$O(n^2 + n)$，$O(n^3)$，$O(n^{100})$はどれも間違ってはいない．

ただし，この場合$O(n^2)$が最も簡潔に精度良く$T(n)$を表現しているので，これを採用する．

これからの授業や多くの教科書でも以上のような前提でビックオー表記を使っています．

空間計算量（領域計算量）

単純には，メモリの消費量.

IoTデバイスのようなメモリが限られる環境や，超大規模なデータを処理する場合などにはよく考える必要あり.

時間計算量とトレードオフになることもある.

アルゴリズムの授業では時間計算量の方が重要視される（ことが多いと思われる）.

そのほかの計算量・計算コスト

通信コストなど.

Computational offloading（モバイルやウェアラブル端末において重い処理をクラウドなどに投げてしまう）などの場合には重要.

時間計算量を体験しよう！

以下のタスクを解くようなアルゴリズムを考えよう.

「ランダムな整数が格納されている長さNの配列の中で,
M個の隣接する要素の和が最大となる部分を1つ求めよ.」

入力：配列（長さN，0以上の整数）とM

出力：M個の隣接する要素の和の最大値と，そのM個の部分列の一番最初のindex.

時間計算量を体験しよう！

例）配列[1, 1, 3, 4, 2]で隣り合う3つの要素の和が最大になるものはどれか？

[1, 1, 3, 4, 2] -> 5

[1, 1, 3, 4, 2] -> 8

[1, 1, 3, 4, 2] -> 9で，これが最大.

ナイーブな考え方

配列の1番目からM番目までを足し合わせ，その値を最大値として記録．indexは0を記録.

ナイーブな考え方

配列の1番目からM番目までを足し合わせ，その値を最大値として記録．和が最大となる場所を保存する変数indexには0を入れておく．

次に，配列の2番目からM+1番目までを足し合わせ，この値が今の最大値より大きければ，最大値とindexを更新．

ナイーブな考え方

配列の1番目からM番目までを足し合わせ，その値を最大値として記録．和が最大となる場所を保存する変数indexには0を入れておく．

次に，配列の2番目からM+1番目までを足し合わせ，この値が今の最大値より大きければ，最大値とindexを更新．

以降，同じ処理をN-M+1番目からN番目の要素の部分和をチェックするまで繰り返す．（部分和が同じになる場合，indexの若い方を優先する．）

実際にやってみると，，，

N=100,000，M=1,000くらいにすると，（私の非力じゃないマシンでも）18秒くらいかかる．

おせー．．．😓

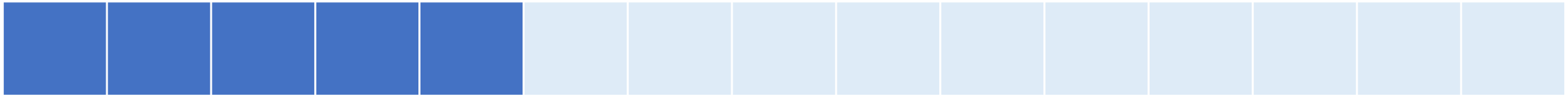
効率的なアルゴリズムの着眼点

無駄を無くそう！

特に，**何度も同じことをやっているのを削減しよう．**

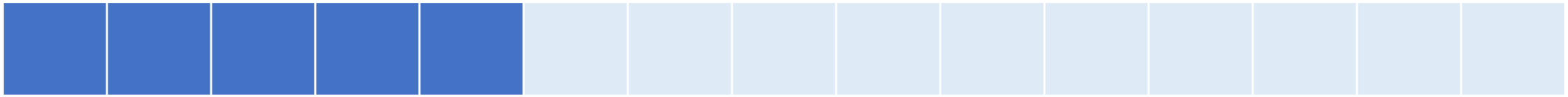
少し深くみてみよう.

ある配列に対して，隣り合う5つの要素の和に対して，最大になるものを求めることを考える.



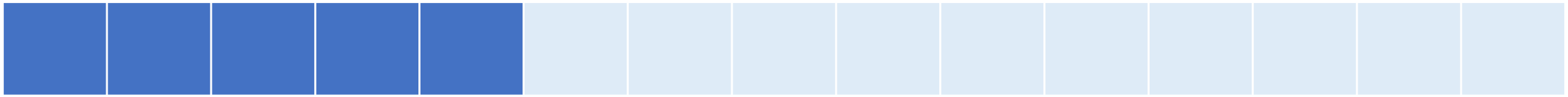
ナイーブな方法の無駄はどこに？

1回目の計算：

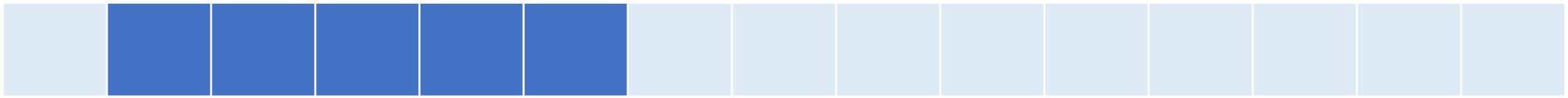


ナイーブな方法の無駄はどこに？

1回目の計算：

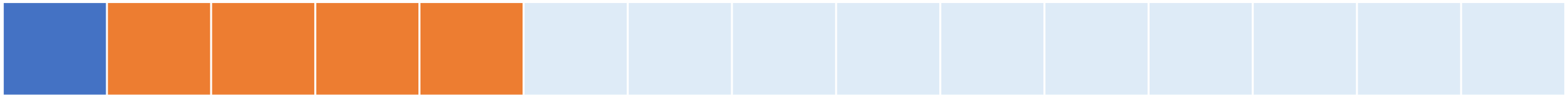


2回目の計算：

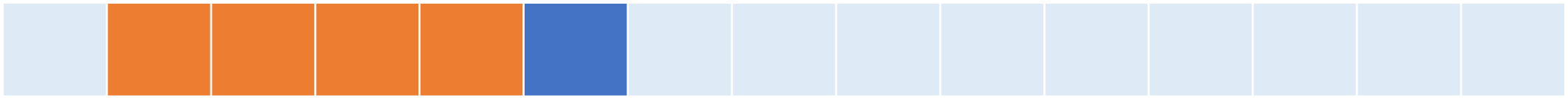


ナイーブな方法の無駄はどこに？

1回目の計算：



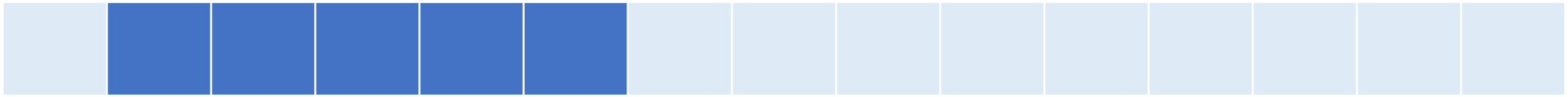
2回目の計算：



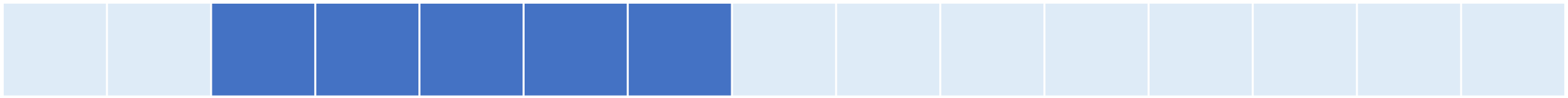
オレンジとのころは全く一緒！

ナイーブな方法の無駄はどこに？

2回目の計算：

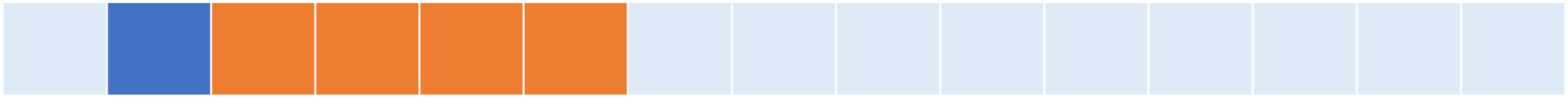


3回目の計算：

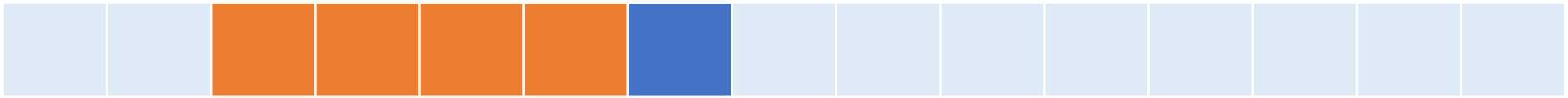


ナイーブな方法の無駄はどこに？

2回目の計算：



3回目の計算：



オレンジとのころは全く一緒！

非効率な部分が見つかった！

毎回の計算で重複する部分があるが，それを毎回再計算してしまっている．

配列の要素に対する更新はないので，毎回計算する必要はないはず．

非効率な部分が見つかった！

毎回の計算で重複する部分があるが，それを毎回再計算してしまっている．

配列の要素に対する更新はないので，毎回計算する必要はないはず．

差分だけうまく計算するにはどうすれば良い？

今日の課題ではこれを考えてみてください！😉

差分だけうまく計算する方式にすると，

実際に実装したものでテストしてみると，N=100,000, M=1,000くらいでも，（さっきと同じマシンで）40ミリ秒くらいで終わる！

やったー！😄

コードチャンレジ：基本課題#1-b [1点]

スライドの説明を踏まえて，「ランダムな整数が格納されている長さNの配列の中で，m個の隣接する要素の和が最大となる部分を1つ求めよ．」の問題を効率的に解くプログラムを実装してください．

実装できたら

自分のローカル環境で，配列の長さやmの値を適当に変化させてみて，計算が完了するまでにどのくらい時間がかかるかを試してみてください．（例えば，配列の長さを10万くらいにするとどうでしょうか？）

ナイーブな方法も実装してみて比較してみてください．どのくらい実行時間が違いますか？

このアルゴリズムの計算量がいくらか，考えてみてください．

ちなみに

ランダムに非負整数が並んだ配列は以下のコードで
作れます.

```
import random

def RandomIntSeq(length):
        seq = random.sample(list(range(0, length)), k=length)
        return seq
```

コードチャンレジ：基本課題#1-a [1点]

「RPGのショップ」

Pythonにおける標準入出力と明示的な型変換を扱う問題.

今後のコードチャレンジにおいてもよく使うと思いますので，今のうちに標準入出力と明示的な型変換ができるようになっておいてください.

開始時に入っている初期コードについて

このコードはtrackで自動的に設定されているものであり，この講義には関係ありません．

この初期コードを使用する必要はありません．（勿論使用した上で解答してもらっても構いません）．

入出力について

コードチャレンジでは標準入力により与えられるデータを受け取り，標準出力で計算結果を表示する必要があります.

出力

標準出力はprintで行うのが最もシンプルかと思います.
（sys.stdoutなどを使ってもらっても勿論OK.）

printを使って出力すると自動的に末尾に改行が入ります.

print実行例

コード

```
x = 1
y = [1, 2, 3]
print(x)
print(y)
print('string’)
```

出力結果

```
1
[1, 2, 3]
string
```

入力

標準入力はinputで行うのが最もシンプルかと思います.
（sys.stdinなどを使ってもらっても勿論OK.）

a = input() とすると，1行分読んでaに代入します.

ただし，文字列として読み込まれることに注意！

input実行例

コード

```
a = input()
b = input()
print(a+b)
```

入力

```
1
2
```

出力結果

```
12
```

入力

必要に応じて型変換を行ってください.

例えば，int型に変えるためには,

```
int(input())
```

とすればよいです.

その他の型変換に関しては各自で調べてみてください.

入力

複数の値が1行に並んでいる場合はどうする？

例）300 20

単にinput()しただけでは，「300 20」という文字列になってしまう．．．

入力

input()で取り込んだ後，スペースで分割し，さらにint型に変換する必要がある．

真面目にやると，

```
a = input()
b = a.split()
M = int(b[0])
L = int(b[1])
```

入力

map関数を使うとより簡潔に！

map関数：配列（リスト）などの各要素全てに対して
指定する関数による操作を一括で適用する.

先ほどの処理は,

```
M, L = map(int, input().split())
```

と1行で記述できます.

入力

複数の値が複数行並んでいる場合はどうする？

例）４８１
10 20 30
100 0 1000
…

map(int, input().split())をループさせて読み込んでいく.
何回ループを回すべきかは事前に与えられるはず.

入力

今回のExtra課題では文字列と数字が入り混じっているので，個別に扱う必要があり．

```
sword 400 10
potion 50 1
spear 250 25
stick 70 1
```

item, *values = input().split()

price, level = map(int, values)

コードチャンレジ：Extra課題#1

今日はExtra課題はありません.

コードチャレンジで使うシステムに慣れておいてください.

やってみよう！

ここから，コードチャレンジで使うシステムの説明.

システムのwalkthroughなどを残りの時間で行います.

コードチャレンジにおけるtrackの利用方法

trackの概要

trackとは

Givery社が提供しているプログラミング学習・試験プラットフォーム

trackとは

エンジニア採用のための試験として使われていることが多い.

今回，特別にGivery様のご協力を得て，この授業のために
システムを利用させていただくことになりました！🤷

trackとは

左側にコーディング問題，右側にコーディング環境が表示され，
ブラウザ上でコーディングを行うことができる．

右下では書いたコードに対するテストが実行され結果を確認できる．

trackの特徴

オンラインエディタ上でコードを書くことができるので手元での
環境構築が一切いらない.

幅広いプログラミング言語に対応（今回の講義ではPython3に
限定していますが）.

その場でテストケースを実行することが出来，自分のコードに対する
フィードバックがすぐに得られる.

本講義でのtrackの利用方法

本講義でのtrackの利用方法

講義があるたびに演習問題を計2〜3題ずつ配信予定.

- 基本課題
- Extra課題

毎回の講義パート終了後，track上での問題ページへのリンクを，
ECCSメールアドレス宛に送信.

track上では，「受験」「試験」などという単語が出てきますが，
試験ではないので安心してください.

課題配信メール

noreply@tracks.runからメールが配信されます．スパムフォルダ等に入ってしまう可能性もあるので，気をつけてください．

デモ配信 Inbox ×

東京大学 矢谷研究室 <noreply@tracks.run>

Japanese > Cebuano Translate message

さま

こんにちは．これはデモの配信メールです．

【試験情報】
試験名　：デモ用
提出期限：2020-03-19 00:00 (GMT+09:00) Asia/Tokyo

【受験手順】
以下のURLから受験を開始してください。
ここに課題提出のためのURLが記載されます．

※モバイルからの受験はできません。次のPCブラウザから受験してください。
(Chrome, Firefox, Safari または Microsoft Edgeの最新版)

課題を行う手順

メールで配信されたリンクをクリックすると課題ページに飛びます.

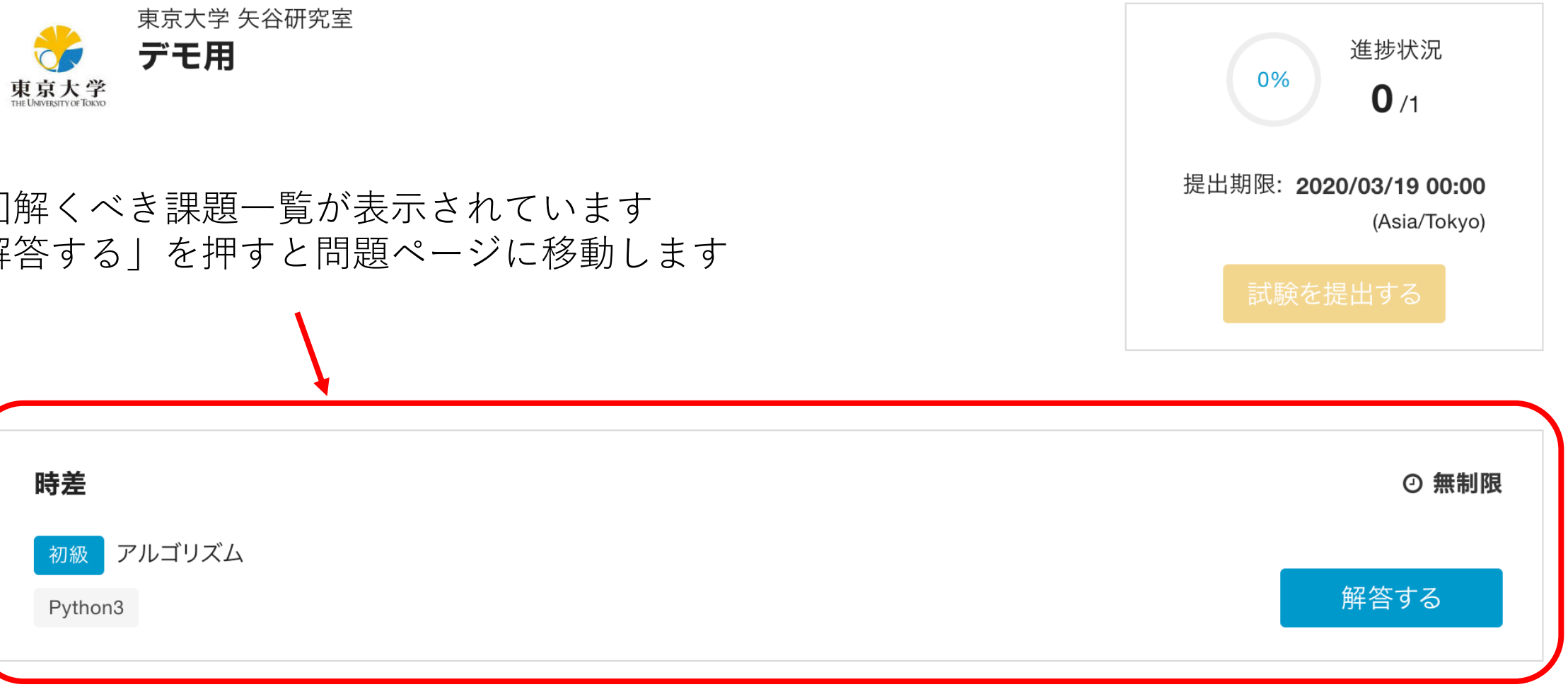
利用規約などに同意してページの一番下の「**試験を開始する**」をクリックすることで課題が見られるようになります.

課題を行う手順

ページが表示されない等の場合，ブラウザを変更するか，adblock等のプラグインを無効化してみてください．

課題を行う手順

試験を開始すると下のような課題一覧ページに飛びます.



今回解くべき課題一覧が表示されています
「解答する」を押すと問題ページに移動します

課題を行う手順 – 問題一覧ページ

Pythonコードを書くエディタ

問題文および入力データのサンプル

ジャッジ結果が表示されるコンソール

課題を行う手順 – 問題一覧ページ

気をつけてほしいこと

与えられた基本課題においては，入出力や問題文で指示がある場合を除き，実装すべき処理において標準ライブラリや問題で指定されていない外部の関数やライブラリ等を使用しないでください．そのような提出物はテストケースに合格していても，採点されませんで注意してください．

Extra課題に関しては，必要に応じて標準ライブラリや外部の関数などを使っても構いません．ただし，必要なものをすべて提出物の中に組み入れるようにしてください．

課題を行う手順 – 問題一覧ページ

気をつけてほしいこと

問題文をよく読んでから課題に取り組んでください.

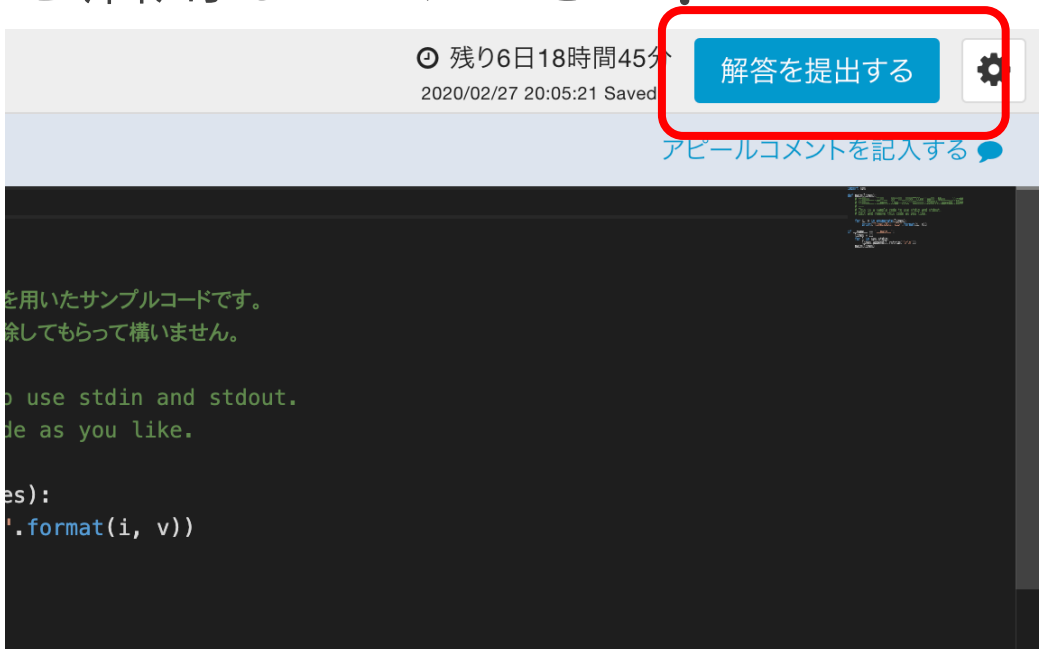
課題によってはコードが一部予め与えられているものがあります.
（track上でデフォルトで与えられるコードと違う場合があります.）
この場合，問題文内で指示がありますので，それに従ってください.
従っていない提出物は採点されないことがあります.

課題を行う手順 – 問題一覧ページ

気をつけてほしいこと

問題ページ右上の**「解答を提出する」を押して一度提出してしまうと再編集ができなくなります．**

作業途中の時は右下の「テストを実行」を押すと，その時点でのコードが自動的に保存されるので，それを活用してください．



課題を行う手順 – コーディング環境について

本講義ではPython3 (Python 3.9)に限定しています.

基本的に標準ライブラリしか含まれていません．状況によって,
numpyなど一部のライブラリは追加するかもしれません.

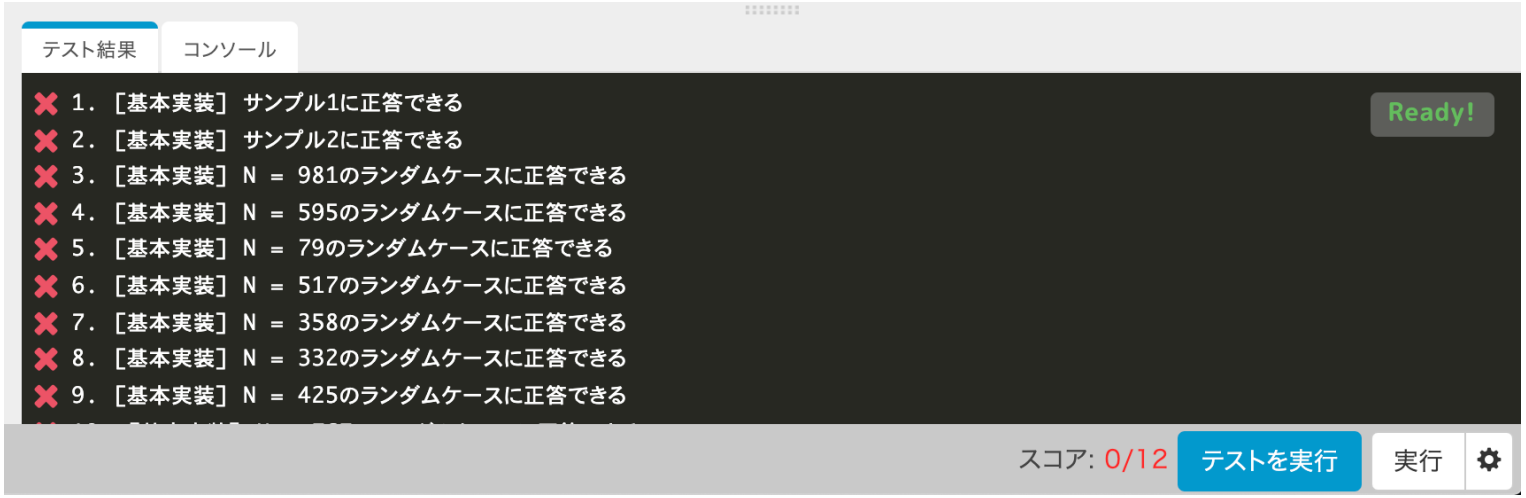
実装・デバッグをできる限りtrack上で行ってください．剽窃等の疑いが生じたときにtrack上でコードの更新履歴を確認することがあります.

課題を行う手順 – ジャッジについて

画面の右下の「**テストを実行**」(または⌘ + Sなど)を押すと用意されたテストケースが実行されます.

右下の「実行」ボタンはこちらでテストケースを自由に設定してコードを実行することができます.

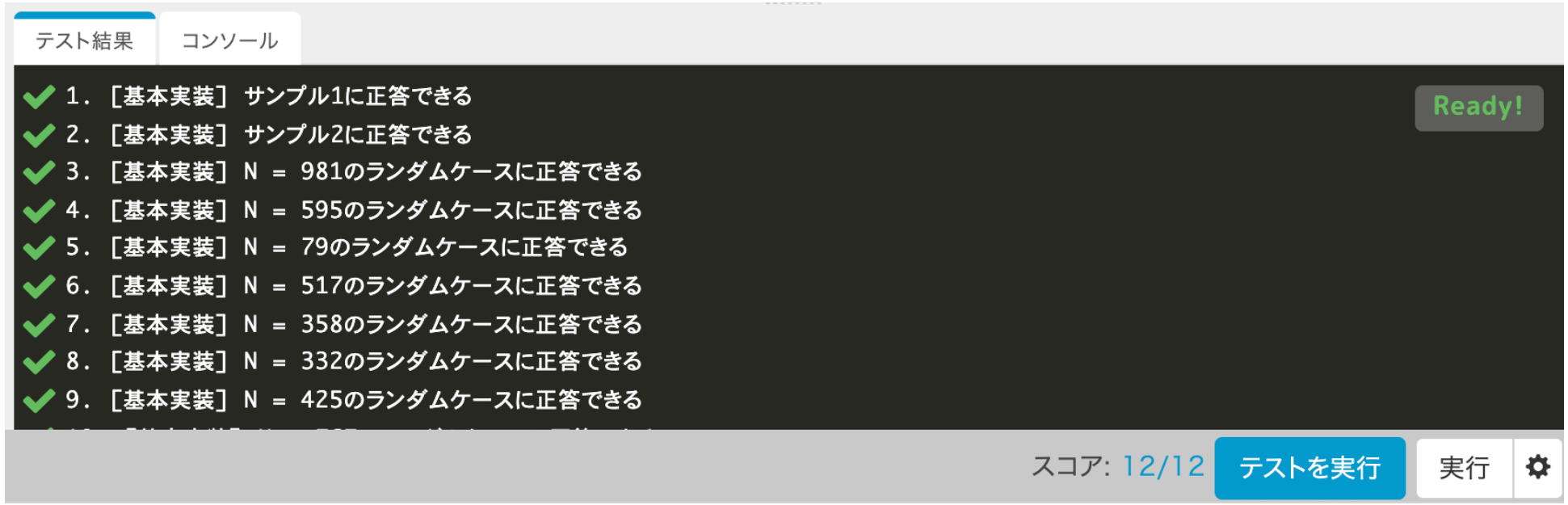
「**テスト結果**」のタブにはそれぞれのテストケースの合否が,
「**コンソール**」のタブには具体的な出力・エラーなどが表示されます.



課題を行う手順 – テストケースについて

各問題について15個程度テストケースが用意されており，通ったテストケースの数に比例して採点が行われます.

テストケースは全て皆さんも確認できるようになっています.



課題を行う手順 – メモリ制限・実行時間

ほぼ全ての問題で

メモリ上限: **512MB**

実行タイムアウト: **5000ms**

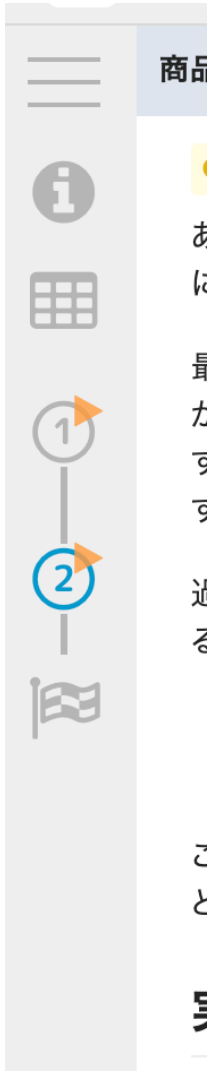
に設定する予定です.

あまりに実行時間がシビアな問題は一部を除いてありませんが,
Pythonには処理が遅い書き方などあるので注意してください.

課題を行う手順 – 問題の切り替え

左のメニューから課題を切り替えることができます.

別の問題を解く際は一度テストを実行することで自動セーブさせておくと安心です.



課題を行う手順 – 解答の提出

それぞれの問題でもうこれ以上変更を加えなくて良い状態になったら，右上の「**解答を提出する**」をクリックしてください．

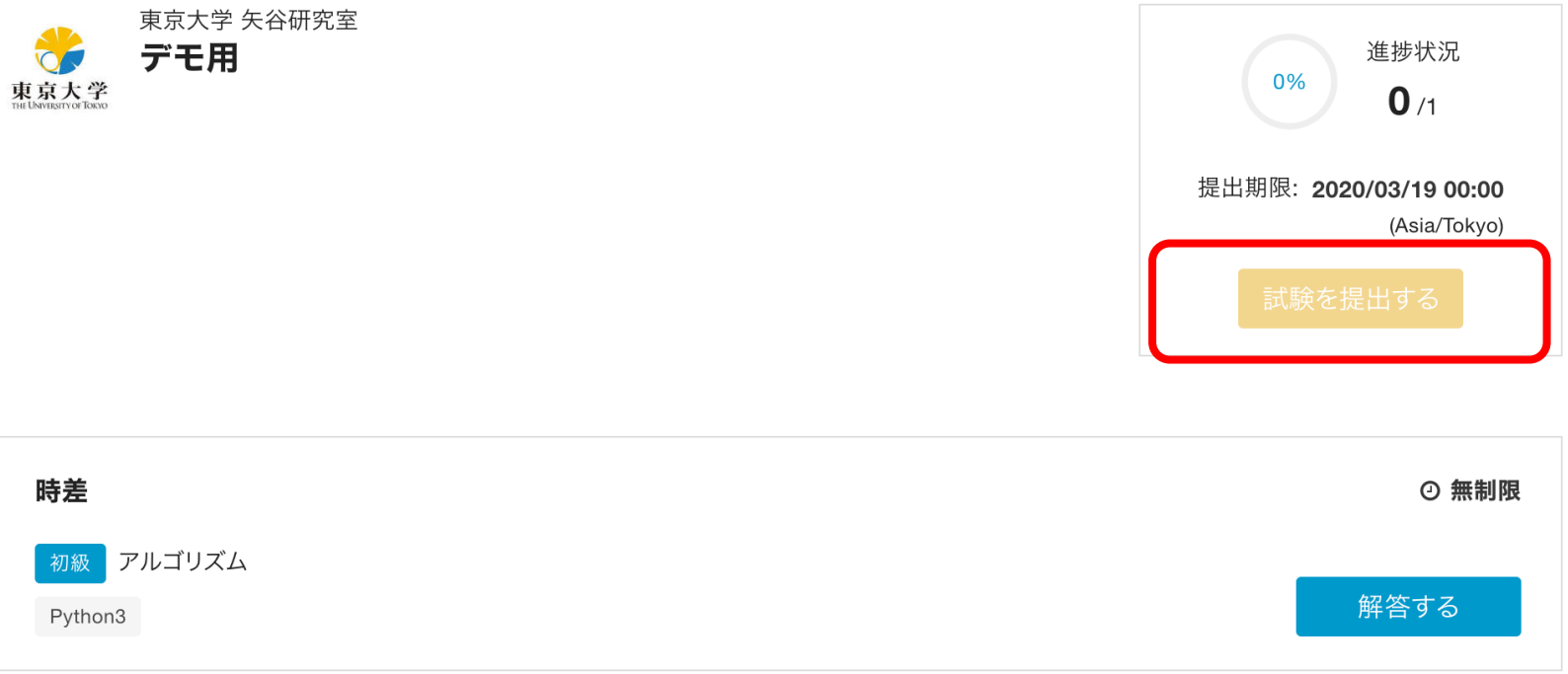
一度提出を完了するとコードの変更はできません．再提出のリクエストは対応できないことがあるので，注意してください．

遷移後，「**ソースコードの説明をする**」画面が出てきます．今回は特に活用する予定はないので書かなくても結構です（何か伝えたいことがある際は書いてもよいです）．

課題を行う手順 – 課題全体の提出

初めの課題一覧ページに戻ると「試験を提出する」ボタンがあるので，
全問解き終えたらこのボタンを押してください．

これを行わずに提出期限が来た場合は，最後に保存された状態のものが
自動的に提出されます．



課題を行う手順 – 課題全体の提出

このイラストが出てくれば課題の提出は完了です． お疲れ様でした！

お疲れ様でした。これで試験は終了です。

もし解答中に深刻な問題が発生した場合は こちら までお知らせください。

課題提出にあたっての注意点

学生さん同士で議論することは推奨します．ただし，コードを直接共有する，などは行わないようにしてください．

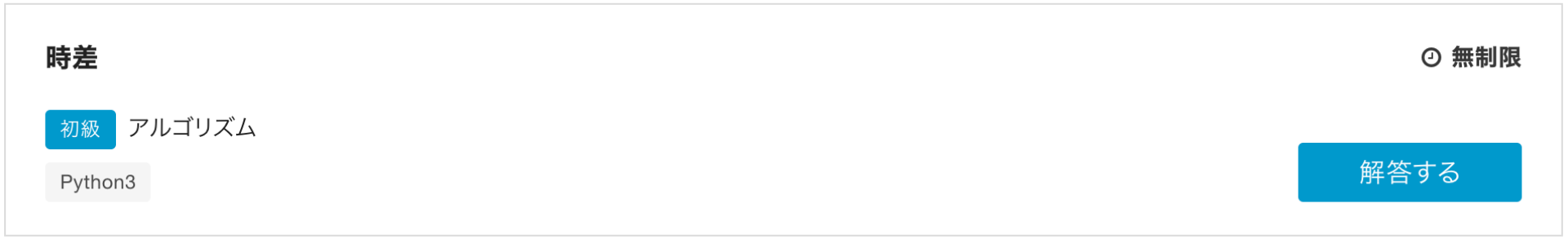
参考書，Webサイト等を適宜参考にしながら，課題に取り組んでもらっても構いません．

TAさんはtrack上のトラブル等には対応しますが，個々のコードのデバッグには手助けできませんので，ご承知おきください．

質問はしていただいてOKですが，課題の性質上答えられないこともあります．

課題提出にあたっての注意点

個々の課題（基本課題，Extra課題）自体には制限時間は設定されていませんが，**全体としての提出期限が設定されています**ので，勘違いしないように注意してください．



課題提出にあたっての注意点

可能な限りtrack上で作業をしてください．**残念ながら自動保存の機能はないので，こまめに保存してください．**

頻繁にテストケースを実行し，デバッグを行ってください．これにより提出間際でのミスを防ぐことができる他，コードが自動保存され，万が一提出作業を行えなくても，最終保存されたコードを元に採点が行われます．

採点は最終提出物に対してのみ行われます．したがって，途中でどれだけミスをしていてもペナルティはありません．

課題提出にあたっての注意点

基本課題に関しては，こちらが事前にインストールや用意しているもの以外のライブラリ等は指示がない限り利用できません．そのような提出物はテストケースに合格していても，無効としますので注意してください．

Extra課題に関しては，外部のライブラリ等を利用しても構いません．その場合でもライブラリに処理の大部分を依存してしまっているコードである場合には得点が無効となります．メインの実装を自分ですることが課題の本質である，とご理解ください．

課題提出にあたっての注意点

提出物に対しては後日類似度チェックを行います．以下のような提出物は該当するものすべてに対して採点を取り消します．

- 提出物間でほぼ同一のコード
- Webや参考書に記載されているものとほぼ同一であることが判明したコード
- Track上での課題取り組み時間（終了時刻 - 開始時刻）が極端に短い
- 与えられたテストケースに通るようにだけ設計されたコード
- 課題で指定されている条件を満たしていないコード
- その他，明らかに不正行為の証拠があるもの

悪質な場合にはより大きなペナルティになることがあります．十分気をつけてください．

課題提出にあたっての注意点

コードチャレンジの締め切り後はシステムの仕様上，提出したコードにアクセスすることができません．後日google drive経由で返却しますが，必要に応じて各自で保存しておいてください．

Extra課題に関しては，解答例と簡単な解説をその次の講義回にて共有します．基本課題は授業のスライドにある情報で実装できるようになっていますので，授業のスライドをよく確認してください．

> 基本課題はスライドや他の教科書等を参考にしてもらえば，
> 実装できるレベルに設定していますので，ぜひ自習をして
> もらえればと思います．

コードの返却

こちらの処理が終わり次第，提出されたコードを返却します．この返却されたコードが採点されたものとなります．

未着手（もしくは未提出）の場合，その課題に関してはコード返却はありません．

テストケースは別途共有しますので，自習にお使いください．

コードの返却

今日の授業終了後，コード返却のテストを行います．以下のようなメールが皆さんのECCSアカウントに届きます．

00-test.pyというファイルに登録フォームにて届け出された学生証番号の下2桁の数字が記載されています．

もし，メールが届かない，ファイルが作成されていない，下2桁の数字が間違っている，などがある場合のみ，矢谷にDMください．