# Algorithms (2024 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！

本学学生のみ：https://forms.gle/yjhQq32V1FemVeWi8

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です．

slackへの招待待ちの数に限りがありますので，slack**への招待を受け取った方はすぐに**slack**に入ってください．**

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．）

# ちなみに，

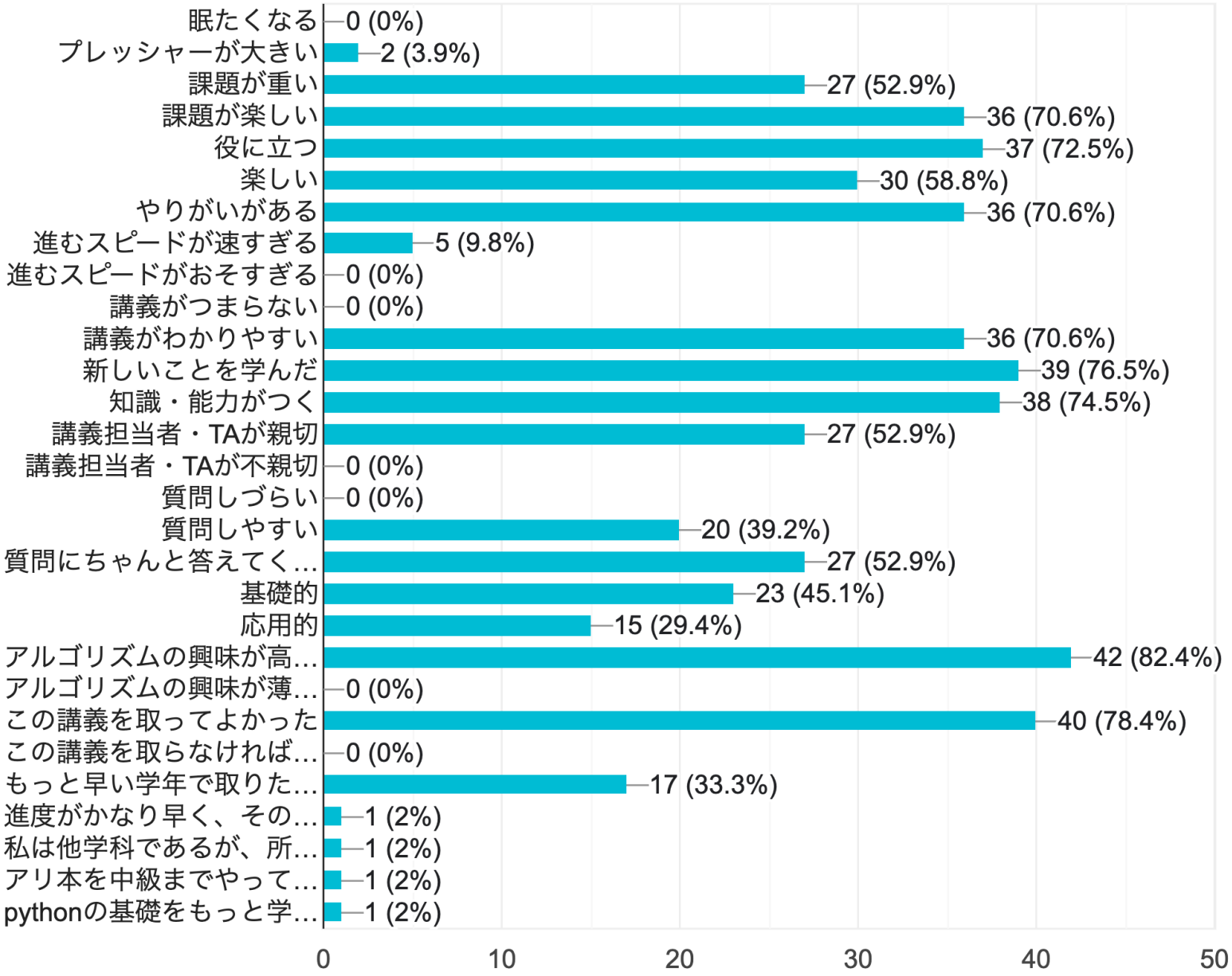
残念ながら，私はアルゴリズムの専門家ではありません．
専門はHCI（ユーザインタフェース）
https://iis-lab.org，https://yatani.jp

ですが，情報技術や情報を人にとってわかりやすい形に設計することには，強い興味＆自信があります．

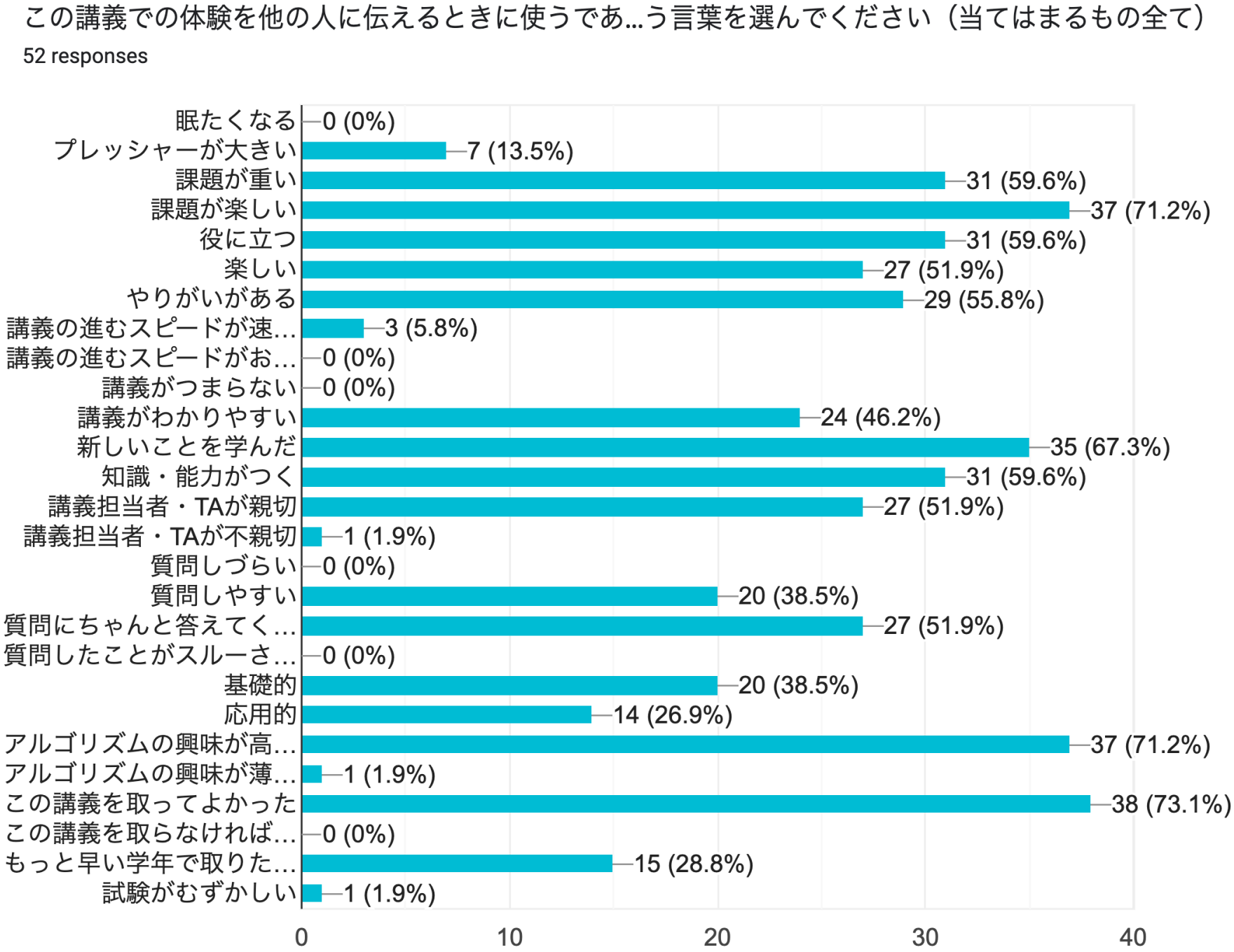
この講義でも「わかりやすさ」を重視してご説明を差し上げていきたいと思いますので，どうぞお付き合いください．

この講義での体験を他の人に伝えるときに使うで...う言葉を選んでください（当てはまるもの全て）

51 responses



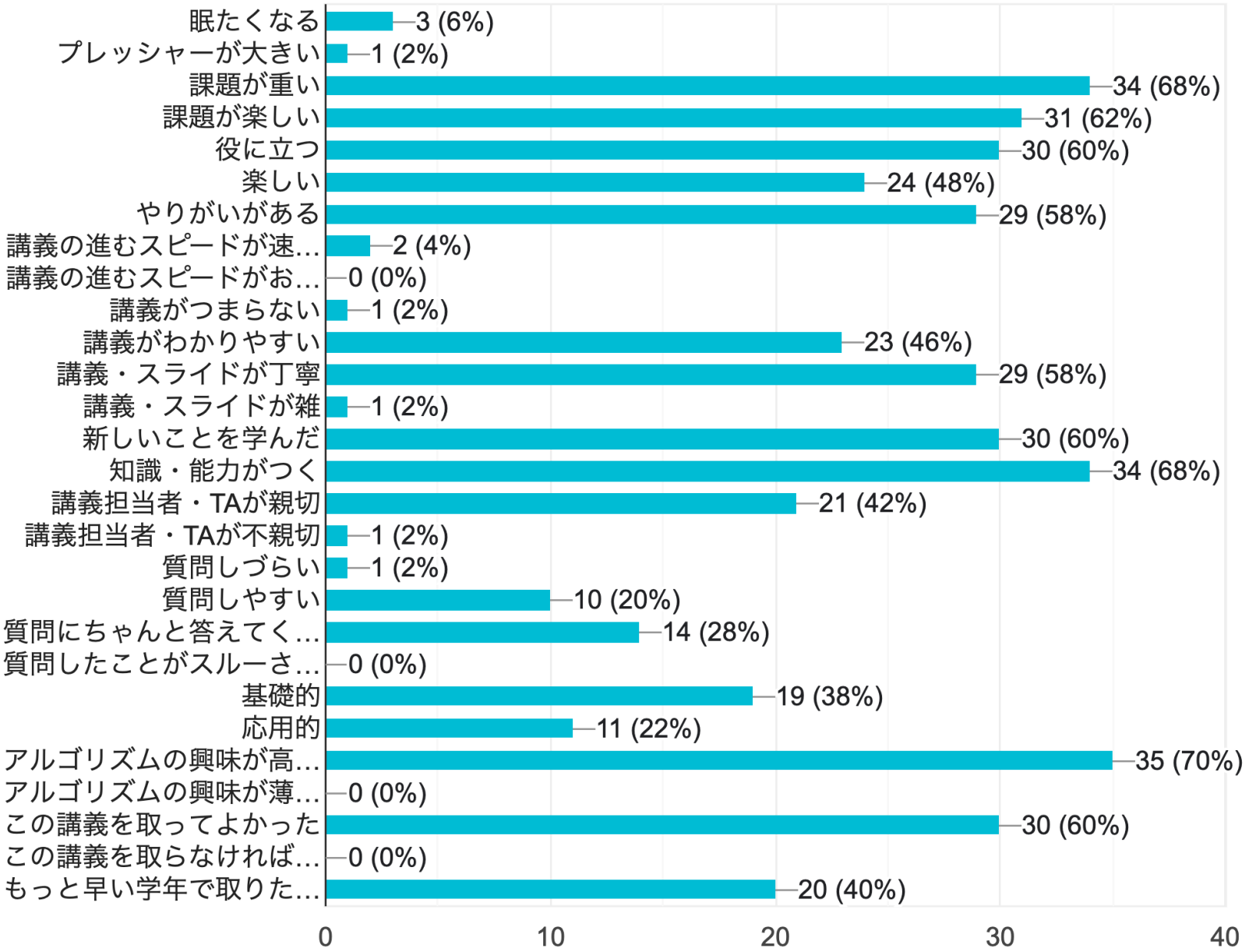

# 2020年度
# アンケート結果




# 2021年度 アンケート結果

この講義での体験を他の人に伝えるときに使うで...う言葉を選んでください（当てはまるもの全て）

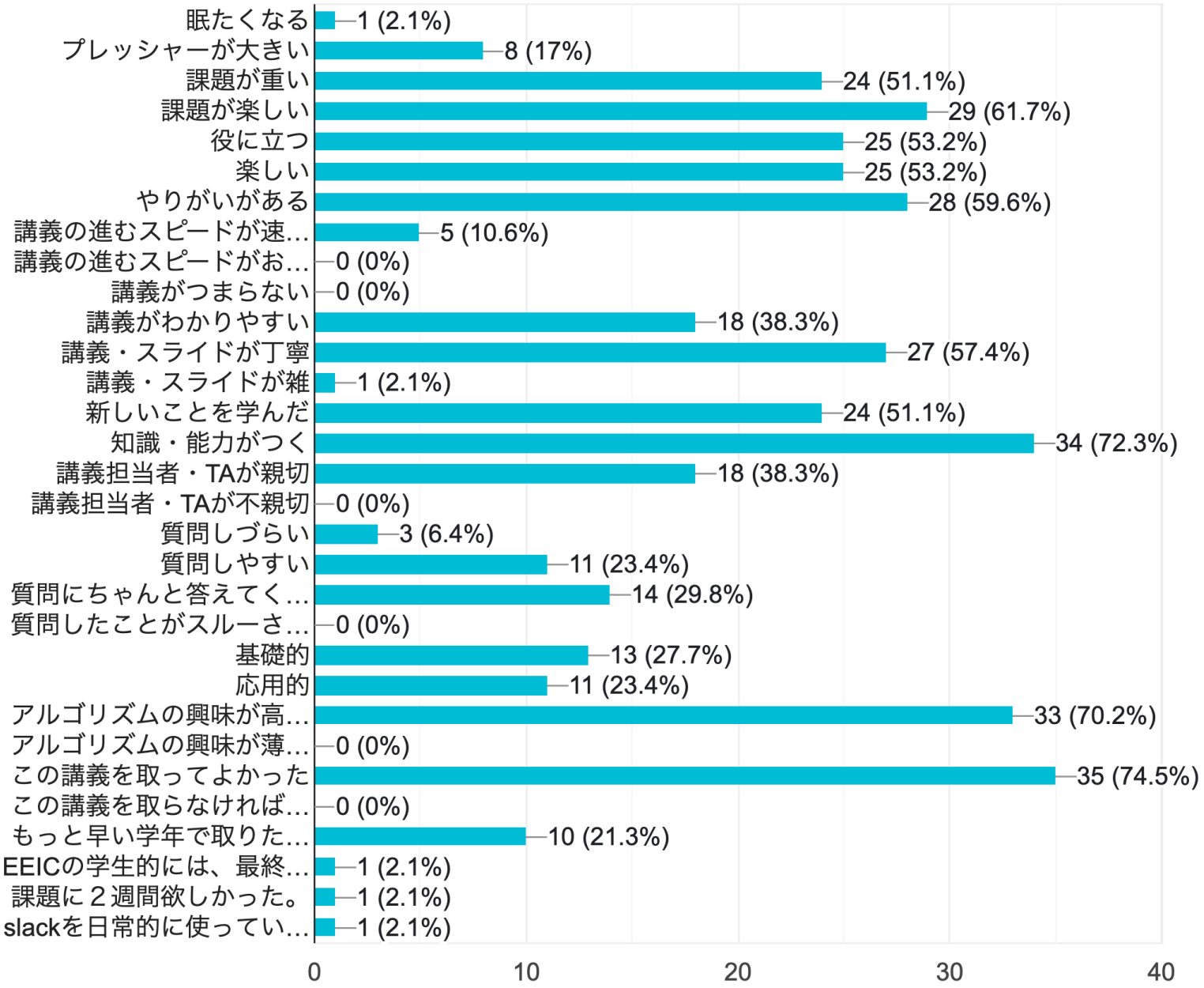
50 responses




# 2022年度 アンケート結果

この講義での体験を他の人に伝えるときに使うで...う言葉を選んでください（当てはまるもの全て）

47 responses




# 2023年度
# アンケート結果

# 講義体制

講義担当者：矢谷 浩司
TA：香取 浩紀，廣澤 佑亮，森上 巧，矢野 祥睦

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．

# 講義体制

メタバース工学部とも相乗りをしています.

この講義は本学学生とメタバース工学部生（社会人）が入り混じる珍しい講義ですので，お互いを尊敬しながら，にぎやかにやって行きたいと思います. 😆

メタバース工学部生の方も遠慮なく，本学学生さんといろいろお話ししていただければと思います.

# 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

本学学生：ECCSアカウント

メタバース工学部生：Google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.

# メタバース工学部生向け基礎チュートリアル

後述するtrackにおいて，「プログラミング(Python) 基礎講座 実践編」をgivery社より提供いただいております．

基礎に不安のある方や，復習されたい方はご利用ください．1ヶ月間の限定利用となります（5/9まで）．

なお，再帰に関してはここに含まれていませんので，別途自習をお願いできればと思います．

# 講義

241講義室，およびYoutube Liveのハイブリッド．

皆さんの顔を見ながら講義をさせてもらうのは，私の喜びでもありますので，可能な限り講義室にお越しください．😊

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．

# Youtube Liveでの配信

Youtube LiveのURLは講義のホームページからリンクしている共有ドキュメントを参照してください.

メタバース工学部生は#metaverseにて共有しているドキュメントを参照してください.

講義開始時刻5分前くらいから，Youtube Liveでの配信を開始します.

講義終了後も同URLで視聴できますので，遠慮なく自習に使ってください.

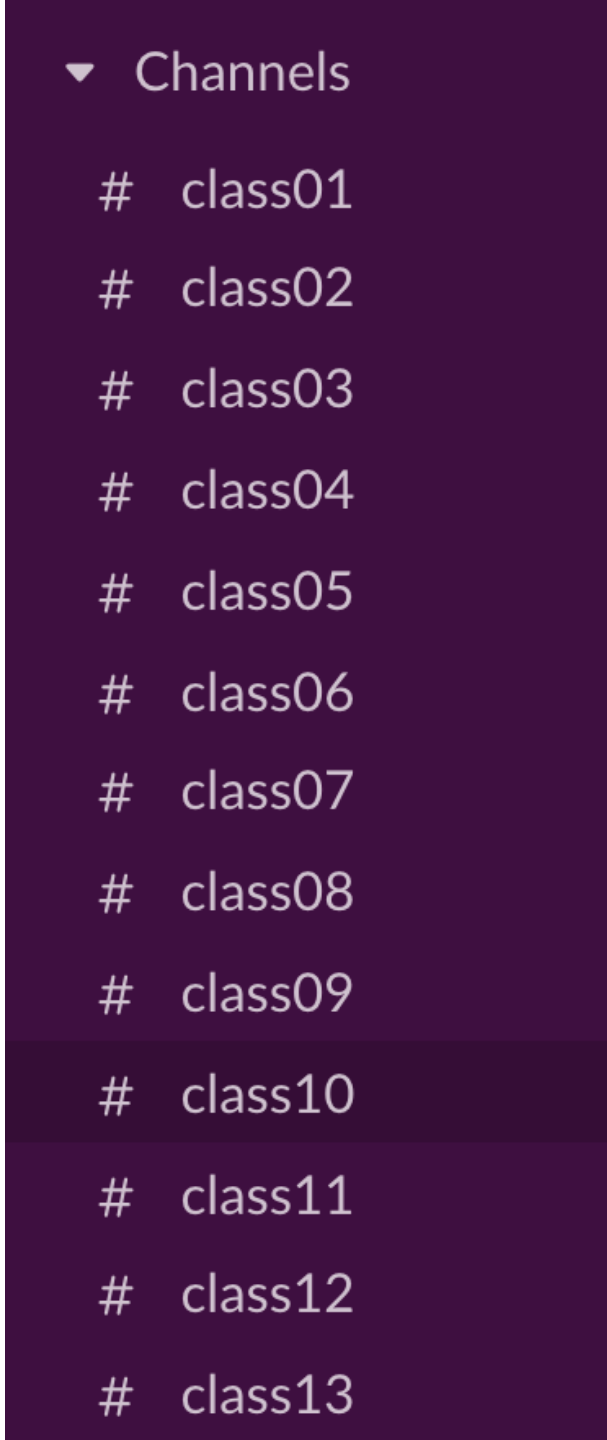
# Q&A, コメント

講義のslack（EEIC-Algorithm2024）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，匿名で投稿できます.

**教室に来ている方もslackでご質問ください.**



# Slack お試し

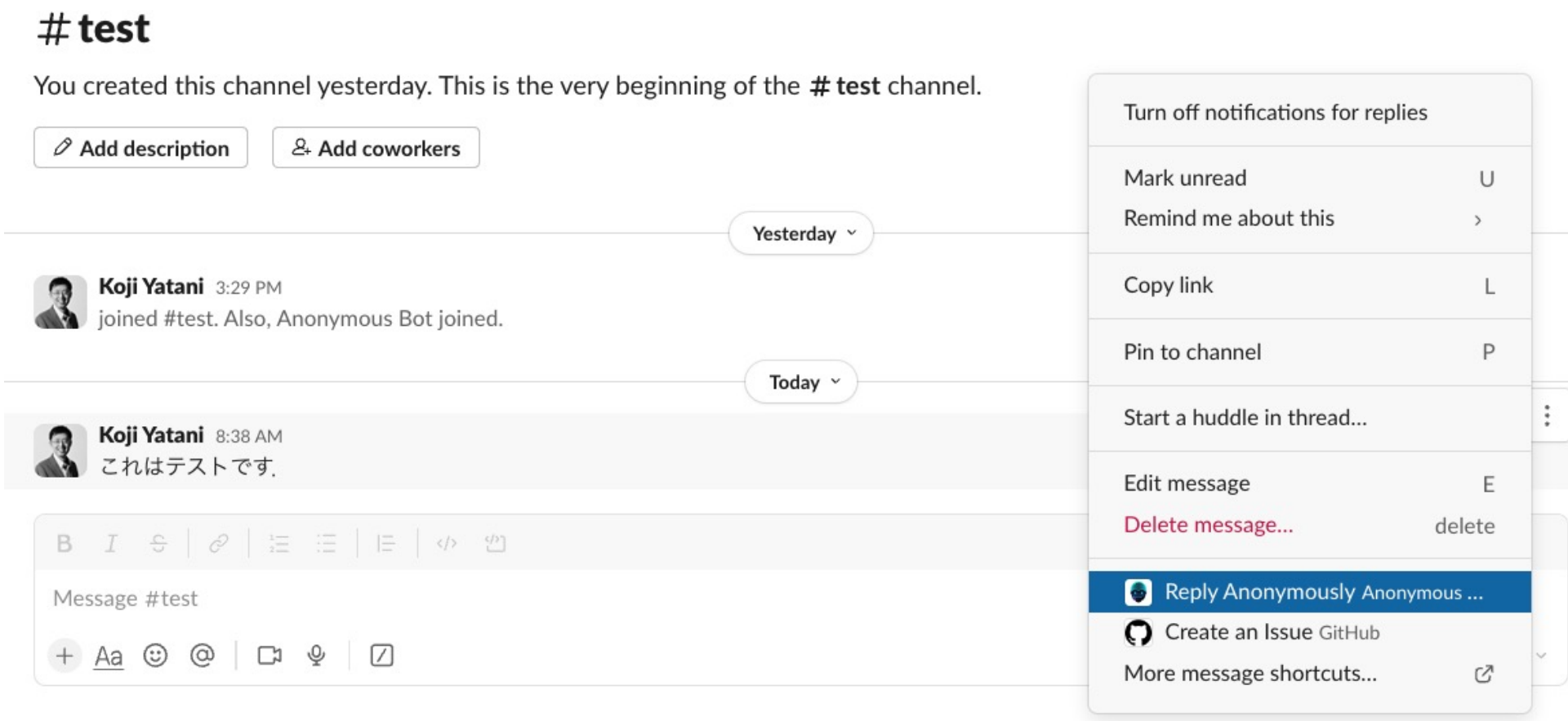
皆さんの今の「推し」を教えてください！😆

以下の要領で，/anonymousを試してみてください.

- #class01に行き，/anonymousとだけ書き，送信する.
- ポップアップが表示されるので，そこにコメントを入れ，”send”ボタンを押す.

# Q&A, コメント

該当するコメントのメニューから，匿名でスレッドへの返信もできます．



# Q&A, コメント

slackでは授業の質問以外にも，課題に関する質問や議論を行っていただいて構いません.

後述の通り，議論に必要な部分のコードを共有することは構いませんが，正答のコードの共有は禁止します.

# コミュニケーションに関して

色々なバックグラウンドの方が本講義を受講してくださっています．異なる意見や文化を尊重し，肯定的なコミュニケーションを行っていきましょう．

言葉の選び方に注意し，他者を不快にさせるような表現は避けましょう．

性的，人種的，またはいかなるハラスメントも厳しく禁じます．ハラスメントを目撃したり、体験したりした場合，すぐに講義担当者にご連絡ください．

# 成績評価（本学学生）

**コードチャレンジ基本課題**（全24課題）：30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1課題）

**期末試験**：37点

ただし，期末試験を放棄（受験しなかった）場合，この講義の成績は「未受験」という扱いになります．

# 成績評価（メタバース工学部生）

以下の条件を全て満たす場合，「修了」となります．

コードチャレンジ基本課題（全24課題，30点満点）：

総得点が15点以上．

0点の提出（未着手のままの提出，着手したが0点の提出，どちらの場合でも）が5回以下．

Omnicampusでの出席アンケート：8回以上の提出．

# 成績評価（メタバース工学部生）

Extra課題は提出しなくても成績に対するペナルティはありませんが，提出された場合は0.5を乗じた上で総得点に加算することとします．

例）基本課題で10点，Extra課題で10点を取った場合，

→ 10 + 0.5 * 10 = 15となり，修了基準の1つを満たすことになります．

# 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.

課題の公開は講義終了の直前に行います.

# 基本課題：全員できてほしい課題

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.

# コードチャレンジの締切

本学学生：

**1週間後の水曜日**0:00（**直近の火曜日の**24:00）

メタバース工学部生：

**2週間後の水曜日**0:00（**直近の次の火曜日の**24:00）

本学学生とメタバース工学部生で締切が違いますので,
十分に気をつけてください.

# コードチャレンジの締切

**「土日も課題に取り組め😈」という意図は一切ありません.**

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はしませんが，しっかりリフレッシュする時間を取ることも意識し，自分なりの生活・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.

ホームページに掲載予定ですので，ご参照ください.
（公開後slackにてアナウンスします．）

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

工学部2号館内の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.

メタバース工学部生の修了要件に，期末試験の受験は含まれません.

# 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

# 出席（本学学生）

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.

# 出席（メタバース工学部生）

別途，メタバース工学部より出席管理（Omnicampus）がありますので，ご対応をお願いします．

Omnicampusに関する質問はslackの#metaverseにてお願いいたします（メタバース工学部事務局の方がご対応いただけるはずです）．

# 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．

# ヘルプが必要な時は

#help-meチャネルに入り，ヘルプが必要な内容を書いてください.

ここでは/anonymousは使わないでください.

TAさんが個別にプライベートチャネルを作り，対応をしていきます.

場合によっては対応にお時間をいただくことがありますので，ご了承ください.

# Academic misconduct

剽窃，無断流用などが発覚した場合，以下のような処置が取られます．これらよりも厳しい処罰が下るケースもあります．

- 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（自動的に不可）．

また，所属先の担当者へも本不正行為を報告します．

# 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.

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
対して処罰を課します.

基本課題に関しては似たようなコードになることは想定されますが,（ほぼ）同一のコードになることは考えにくいため，同一性の高いコードに対しては処罰を課します.

# 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.

不正が疑われる場合には，過去のバージョンに遡ってコード行数の増減や，テストケースのパス回数を調査し，講義担当者とTA4名が全員で総合的に判断をします.

# 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．**

# Academic misconduct

この講義にいても同一のコードの提出があり，関係した学生に対しても調査を行った結果，コードの共有が確認されたケースがありました.

共有されたコードを提出した学生のみならず，**コードを共有した学生にも処罰を課す**ことになりました.

不正行為への加担・幇助も処罰の対象となりますので，甘い考えでコードの共有を行わないようにしてください.

# 生成AIの利用

生成AIや自然言語処理・人工知能サービスを利用してコードを生成し，その全部，もしくは一部を利用して回答した場合，**本講義においては即座に不正行為とみなし，今までに提出されたすべての課題を不採点，かつ以降の課題提出を認めない（自動的に不可）ものとします．**

本やWebの情報をそのまま使うのと同じく，自分のために全くなりませんので，そのようなことに無駄に時間を使わないようにしていただければと思います．

# 外部リソースの活用

ただし，生成AIや自然言語処理・人工知能サービス，あるいはWebで調べた内容や説明，解説のうち，自分の理解に大きく役に立ったものがあれば，ぜひ講義の各回のチャネルで共有してください．

**ただし，この場合も，コード例をそのまま貼り付けるなどはしないようにしてください．**

# Let’s help each other!

課題等でわからないことがあれば，講義の各回のチャネルで共有してください.

私やTAさんのほか，学生の皆さんもできる範囲内で積極的にサポートしてもらえれば嬉しいです.

コメントを返すのを特に目立って行っている方には，別途成績で考慮したいと思います.

# ご登録をお願いします！

本学学生のみ：https://forms.gle/yjhQq32V1FemVeWi8

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です．

slackへの招待待ちの数に限りがありますので，slack**への招待を受け取った方はすぐに**slack**に入ってください．**

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．

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（あくまで感覚）

$n = 10^6$

$10^8$ステップ/秒

| | |
|---|---|
| $O(n^2)$ | 2.7時間 |
| $O(n \log n)$ | 200ミリ秒 |
| $O(n)$ | 10ミリ秒 |

# オーダーの感覚（あくまで感覚）

$n = 10^6$ 2.7時間

$O(n^2)$

$n = 10^5$ 100秒

$10^8$ステップ/秒

$n = 10^4$ 1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，アルゴリズムの計算量の話では，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，以下はどれも間違っているわけではない.

$O(n^2)$，$O(n^2 + n)$，$O(n^3)$，$O(n^{100})$，$O(2^n)$

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.

この授業や多くの教科書でも以上のような前提・約束でビックオー表記による説明を行っています.

この授業の試験においても同様.

# 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index 0からM-1までの要素を足し合わせ，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．

# ナイーブな考え方

配列のindex 0からM-1までの要素を足し合わせ，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index 1からMまでの要素を足し合わせ，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

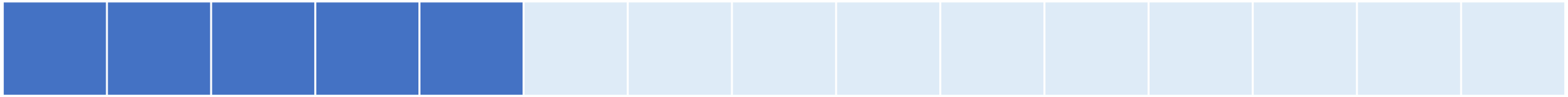
配列のindex 0からM-1までの要素を足し合わせ，その値を最大値として記録．和が最大となる場所を保存する変数indexには0を入れておく．

次に，配列のindex 1からMまでの要素を足し合わせ，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index N-MからN-1まで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．）

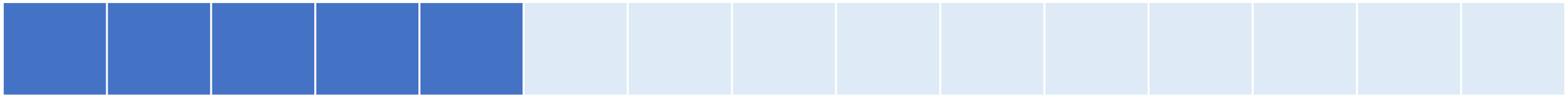
# ナイーブな考え方を図示すると

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．

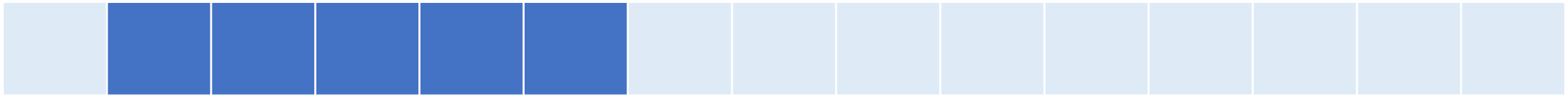


# ナイーブな考え方を図示すると

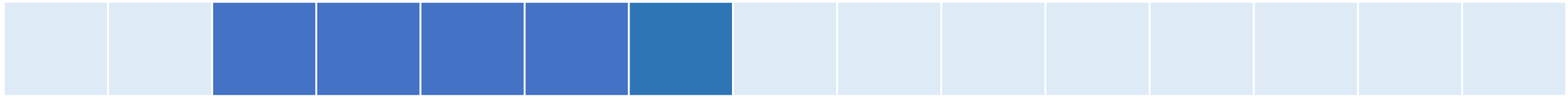
1回目の計算：



2回目の計算：



3回目の計算：



# 実際にやってみると，，，

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

おそい．．．😓

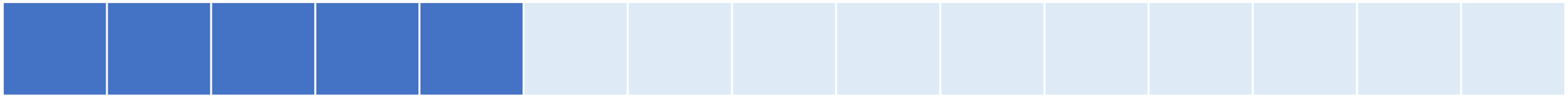
# 効率的なアルゴリズムの着眼点

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．**

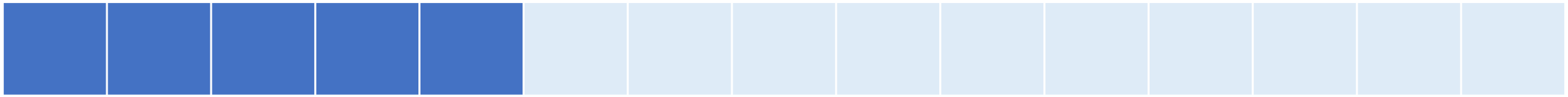
# ナイーブな方法の無駄はどこに？

1回目の計算：

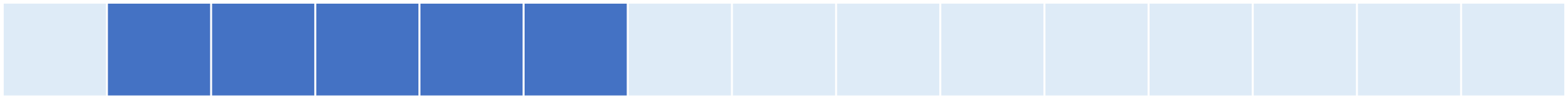


# ナイーブな方法の無駄はどこに？

1回目の計算：

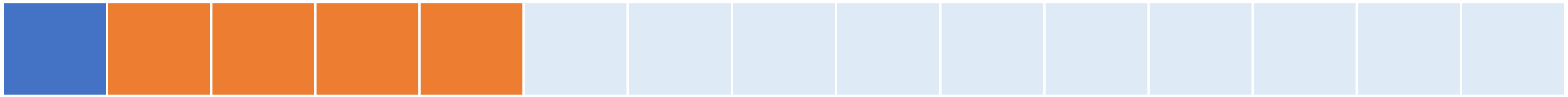


2回目の計算：

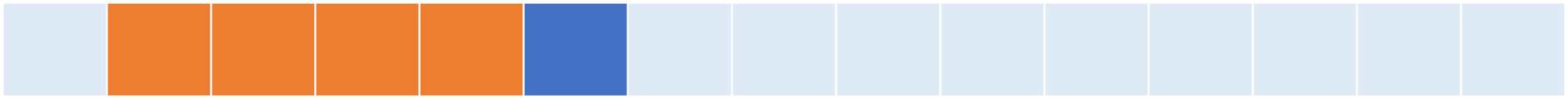


# ナイーブな方法の無駄はどこに？

1回目の計算：



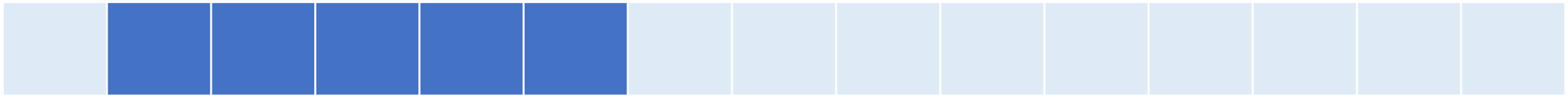
2回目の計算：



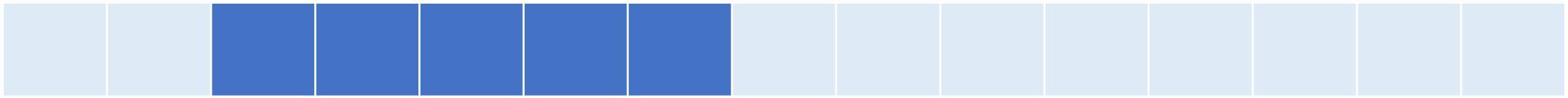
オレンジとのころは全く一緒！

# ナイーブな方法の無駄はどこに？

2回目の計算：

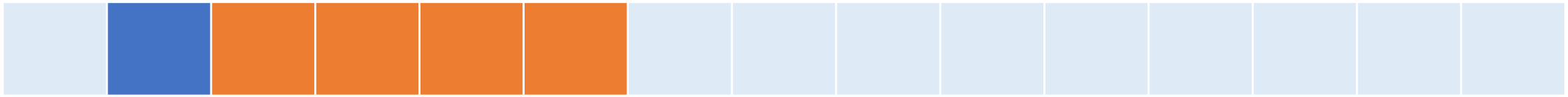


3回目の計算：

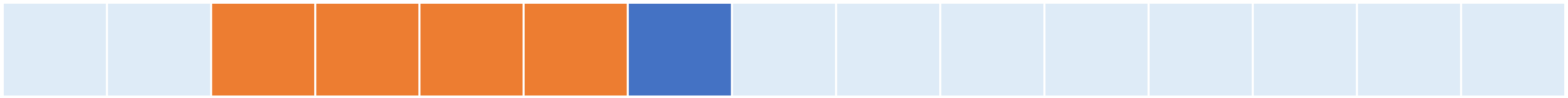


# ナイーブな方法の無駄はどこに？

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
```

# コードチャレンジ：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

基本課題1-a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)

みたいにするとよいです.

# コードチャレンジ：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.

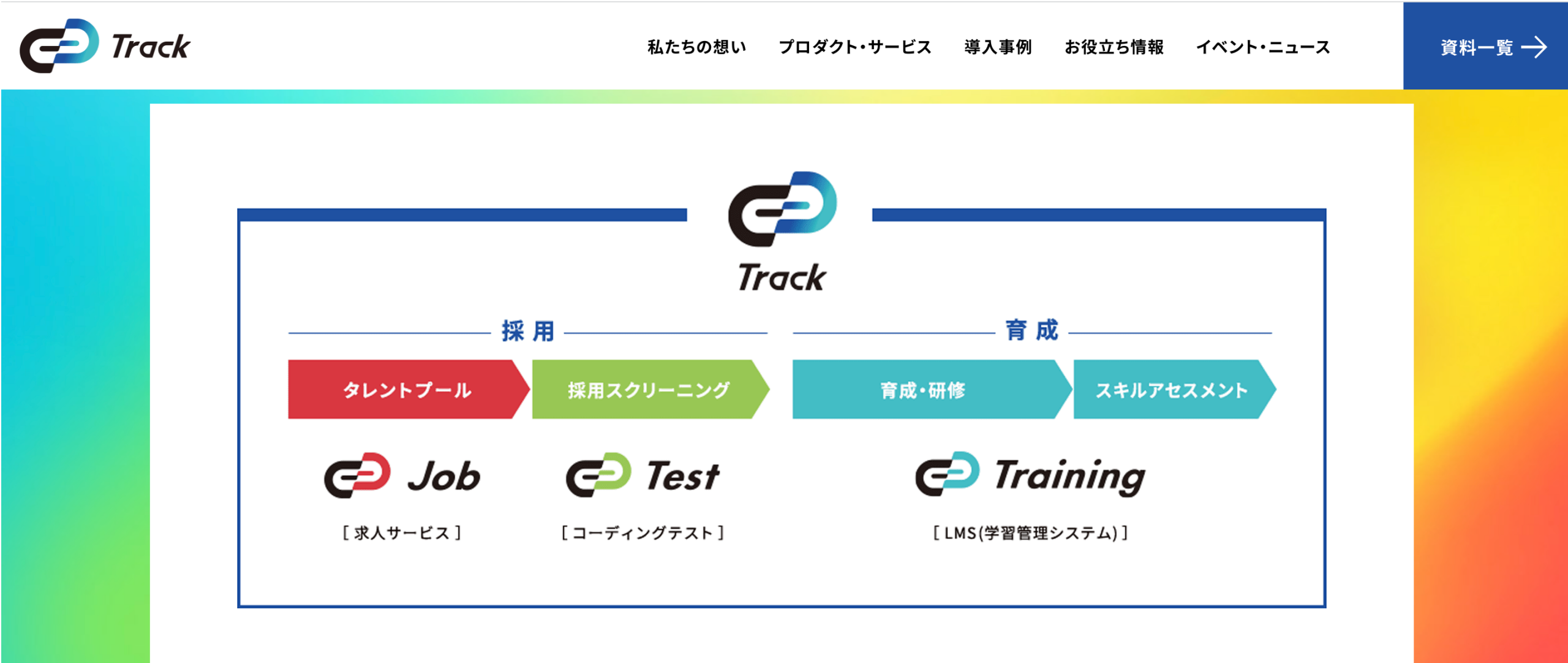
システムのwalkthroughなどを残りの時間で行います.

# 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上での課題を公開します.

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.

# 課題を行う手順

https://eeic-algorithms.train.tracks.run/auth/login
にアクセスしてください.

アカウント名，初期パスワードは以下のように設定されています.

- 本学学生
  - アカウント名：ECCSアカウントのメールアドレス
- メタバース工学部生
  - アカウント名：ご登録時に使用したGoogleアカウントのメールアドレス

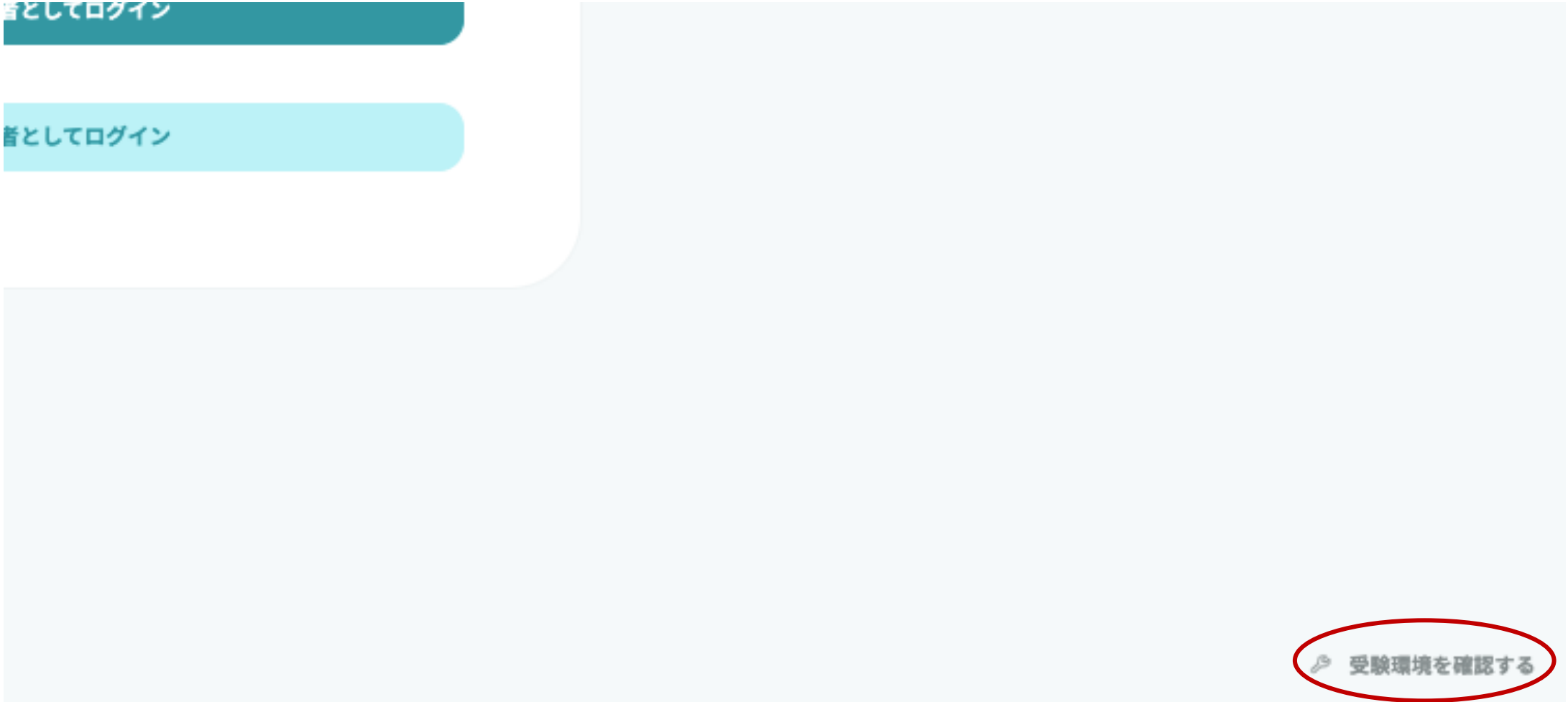
パスワードは届いたメールの指示に従って設定してください.

# 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．**

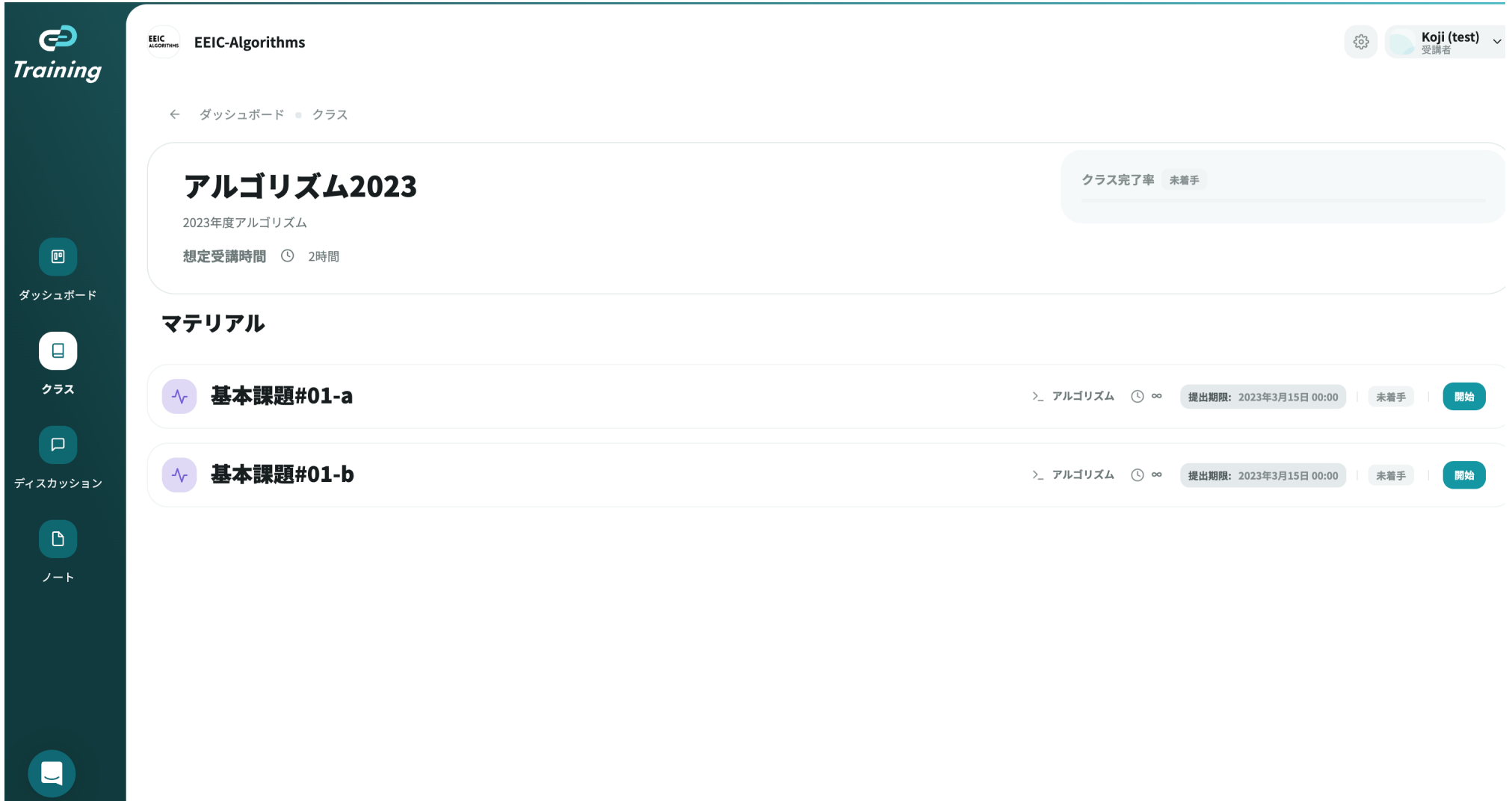
# 課題を行う手順

**右下にある「受験環境を確認する」を押し，すべての項目で問題なしになることを事前に確認してください．**



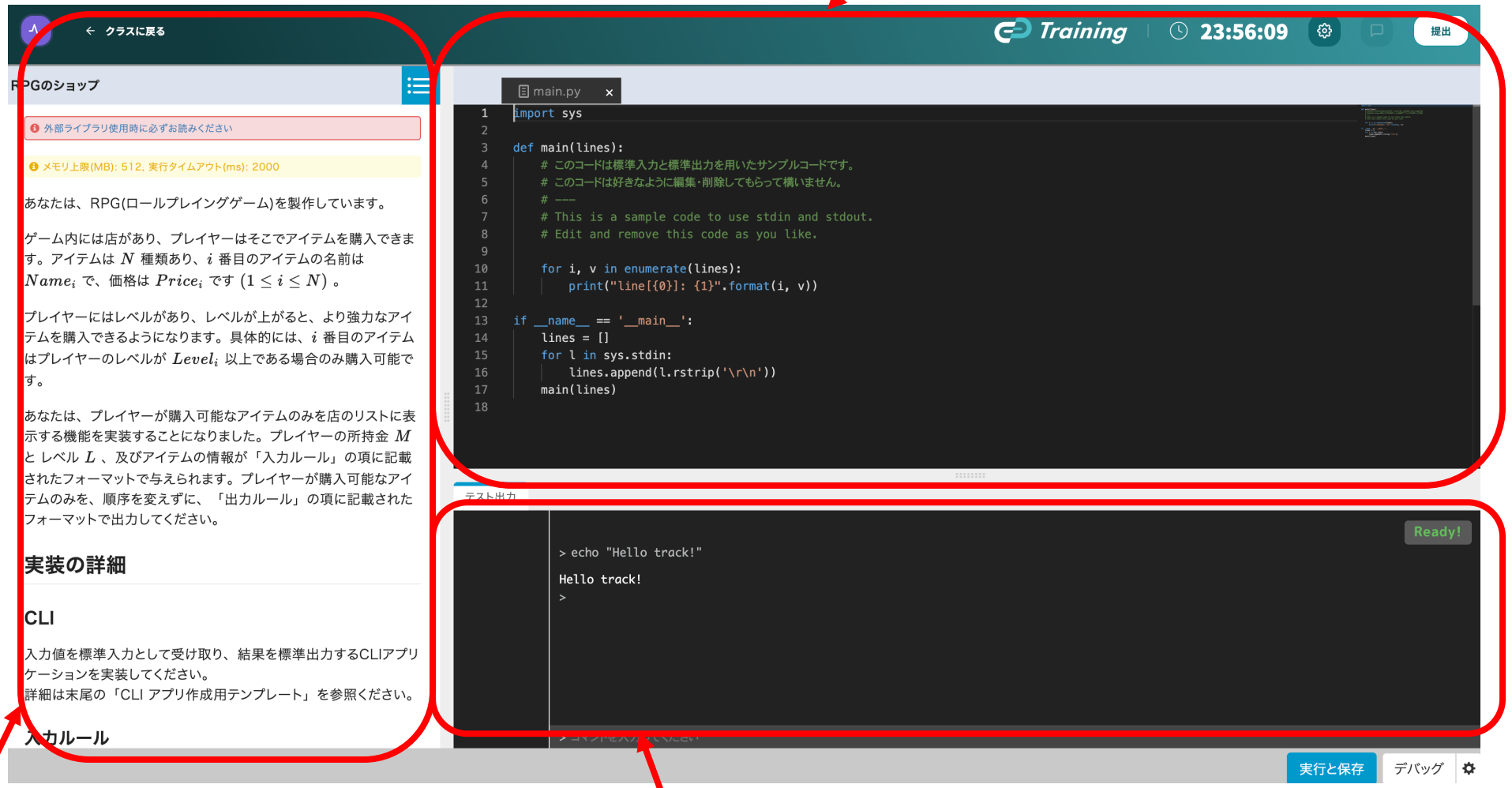
# 課題を行う手順

この講義のクラスをクリックすると課題一覧ページに飛びます.



# 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．また．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.

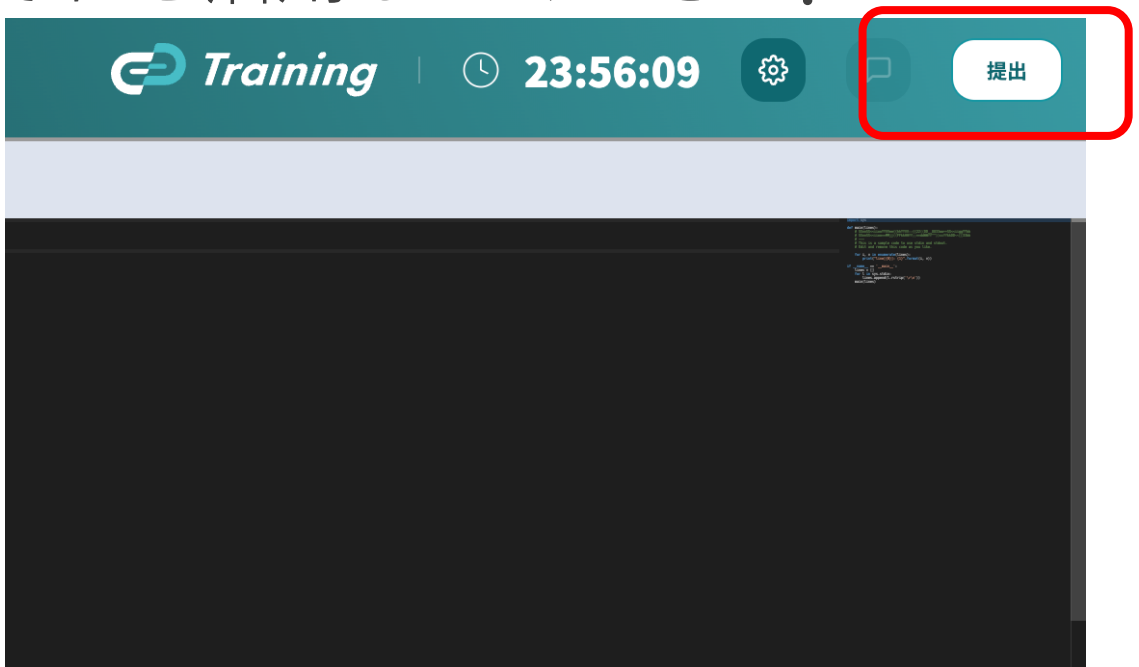
**課題によってはコードが一部予め与えられているものがあります.**
（track上でデフォルトで与えられるコードと違う場合があります.）
この場合，問題文内で指示がありますので，それに従ってください.
従っていない提出物は採点されないことがあります.

# 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**「提出」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.

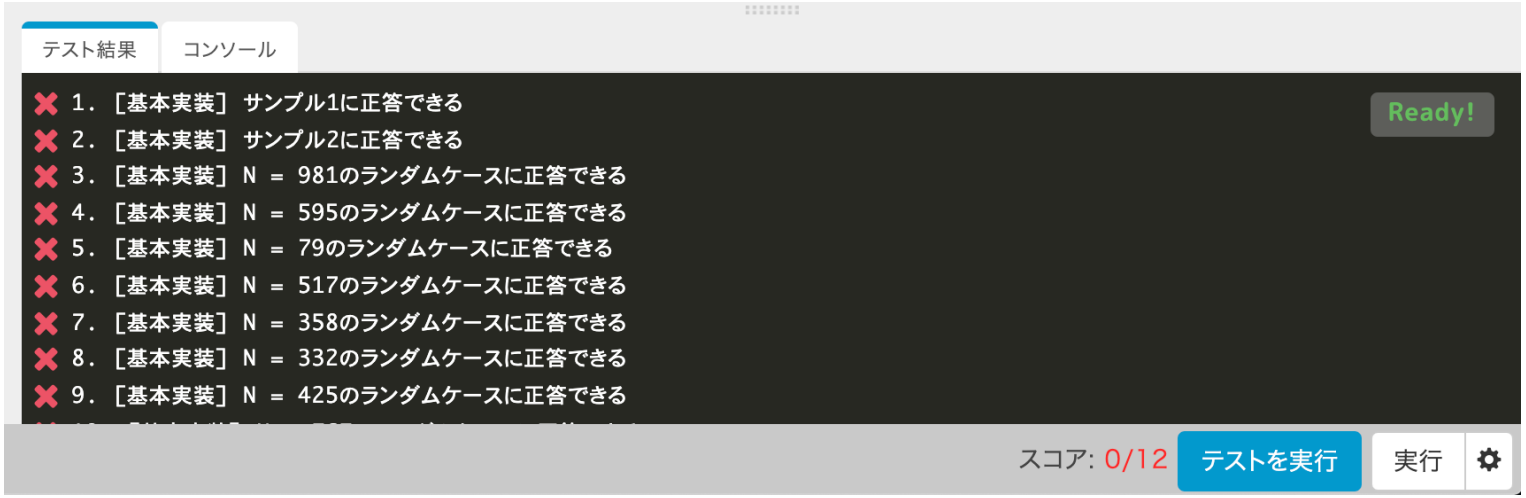
実装・デバッグをできる限りtrack上で行ってください．剽窃等の疑いが生じたときにtrack上でコードの更新履歴を確認することがあります.

# 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.

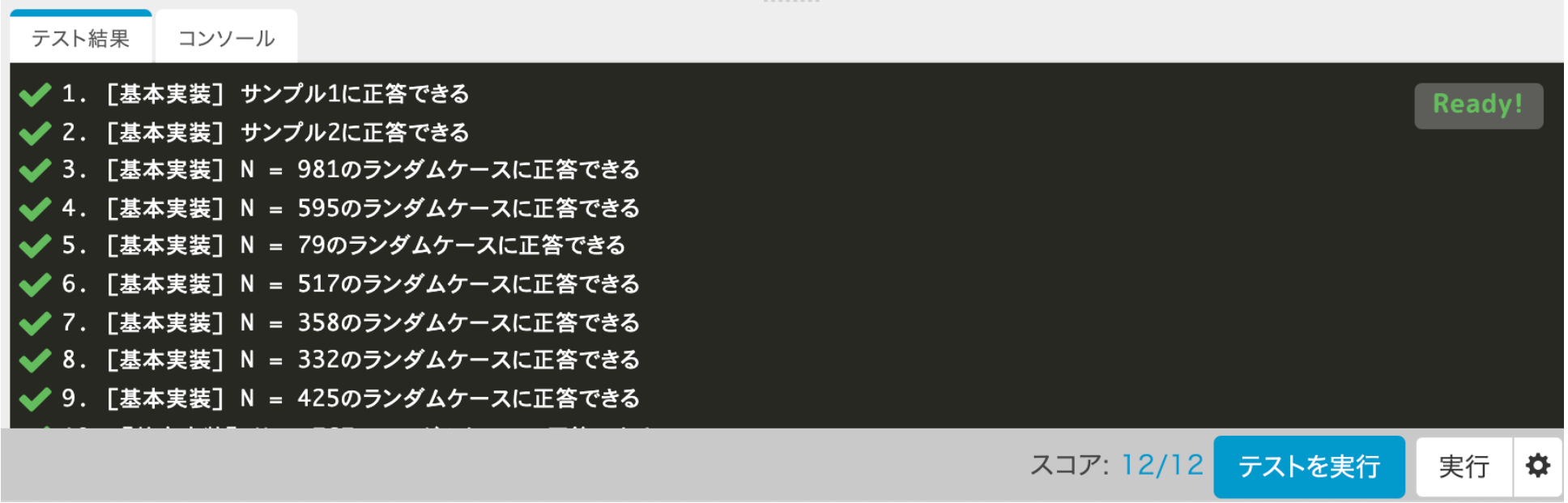
「**テスト結果**」のタブにはそれぞれのテストケースの合否が,
「**コンソール**」のタブには具体的な出力・エラーなどが表示されます.



# 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 – 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「**提出**」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．

# 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

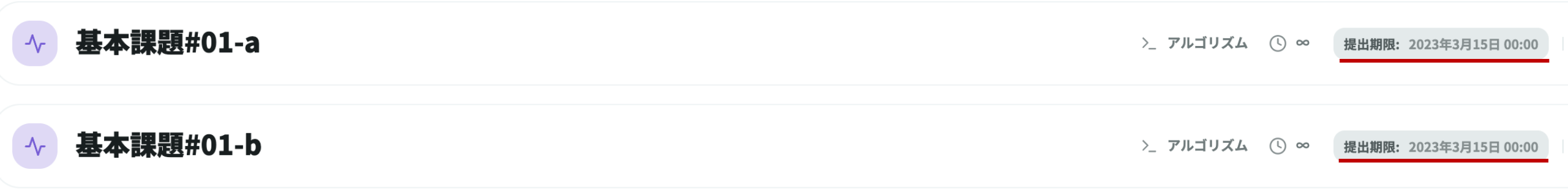
**生成AIによってコードを作り出し，その全部，もしくは一部を用いて課題提出を行なった場合，この講義では不正行為とみなします.**

重大な結果を招くことになりますので，不正行為を行わず，実直に課題に取り組んでください.

# 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，**提出期限は設定されています**ので，勘違いしないように注意してください．

**マテリアル**



# 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（テストケースの実行）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，テストケース合格率が最も高い最新のコードを元に採点が行われます．

採点は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

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．

# コードの返却

提出後も自分のコードをtrack上で見ることが可能です.

Extra課題に関しては，後日簡単な解説と正答例もtrack上で提供します.

また，自習のためにtrack上で再度自分のコードをテストすることも可能です．ただし，返却後のコードは再採点されません（返却後に点数が上がっても，その点数は成績には反映されません）.