

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

クイズ1:「プログラム」という言葉から 連想するものは？

- 授業メモ

- パソコンでなにかカタカタ打ってる。システムを作ってるイメージ。
- 英語がたくさん並んでるイメージ。
- 自分の作りたいものをパソコンに打って、パソコンに実行してもらおう。

「プログラム」という言葉から連想するものは？

- 出典: スーパー大辞林
 - (1) 物事の予定。行事の進行についての計画。
 - (2) 映画演劇コンサートなどの演目や曲目, あらすじや解説などを書いた表や小冊子。
 - (3) コンピューターに, 情報処理を行うための動作手順を指定するもの。また, それを作成すること。
- 授業「プログラミング」でやるのは(3)。でも、(1)～(3)は何かしら共通してるところがあるから「プログラム」と呼ばれているのでは？

行事式次第を例に考えてみよう

1. 2019/4/3

1. 学部長挨拶

2. コース別オリエンテーション

1. コース長挨拶およびコース紹介
(コース長)
2. コースの教員紹介(全教員)
3. 新入生の自己紹介
4. 懇談・質疑応答

3. 教職説明

4. 数学プレースメントテスト

5. インストール大会

上司からの指示

「このプログラムに沿ってオリエンテーションを開催してね。期待してるよ😊」

クイズ2: あなたならどう開催する？

皆さんの考える開催方法

- 授業メモ

「行事の開催」と「プログラムの実行」における 共通部分と差異

行事式次第の開催(実行)

- 書かれている順番通りに司会者が実行する。
- 司会者は人間であり、式次第は自然言語で記述されている。
- 式次第をどう実行するかは、司会者の裁量に委ねられている。
- 実行する度に何かしら違いがある。

プログラムの実行

- 書かれている順番通りにコンピュータが実行する。
- コンピュータは計算機であり、プログラムはプログラミング言語で記述されている。
- プログラムをどう実行するかは、言語仕様で厳密に規程されている。
- 何度実行しても必ず同じように実行する。

「プログラム」の特徴

• 実行するのはコンピュータ

- コンピュータ言語(プログラミング言語)
- Machine code (機械語)
- Assembly language (アセンブラ)
- Basic, C言語,,,,
- Java (後期授業「プログラミング2」)
- 軽量プログラミング言語
 - 何らかの実際の機能によるカテゴライズではなく、習得・学習・使用が容易な言語
 - Perl, PHP, Ruby, Python,,,,

• 再現が容易

- 書いた通りに動く

• 複製が容易

- 複製コストはほぼ無視できる

• (条件付きで)編集や再利用が容易

- テスト(Testing)
- バージョン管理

プログラミングとは？（広辞苑）

- 広義

- コンピューターのプログラムを作成すること。プログラムの仕様の決定、誤りの修正などの作業などを含めていうこともある。

- 狭義

- (仕様通りに)プログラムを書くこと。
- 「コーディング」

- 仕様

- (1) やりかた。方法手段。「返事の一が気に入らない」
- (2) →仕様書に同じ。

- 仕様書

- (1) やり方や、その順序を記した文書。「作業の一」
- (2) 建築・機械などで、注文品の内容や図などを書いた書類。

- 当面は仕様決定済みの課題に取り組む。

プログラミングに含まれる3ステップ

- (コンピュータはプログラミング言語しか知らないので、)プログラマは、実現したいことをプログラミング言語に翻訳する必要がある。
 1. 「実現したいこと」の理解が大前提。この理解が不十分の場合、何を翻訳したら良いかが分からない。
 2. 「理解したこと」を手順として説明できるレベルまで整理する。
 3. 最後に、プログラムへ「翻訳(記述)」する。
 - 初学者は、プログラミング言語の構文・仕様を学ぶ必要がある。

一般的に、「プログラム」には3番目の翻訳結果しか残らない。1の理解や、2の整理が欠落してしまう傾向にある。レポートを書く時、また友人らと一緒に課題に取り組む際には、1,2の説明をするように心がけよう。

プログラミングを円滑に進めるために

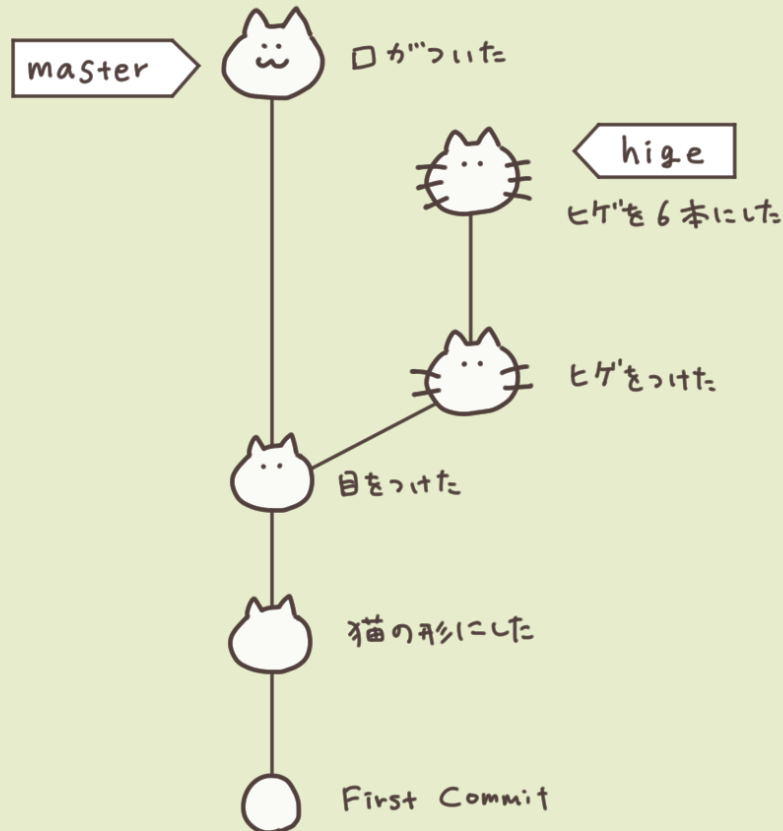
授業計画:
第15回

授業計画:
第6回～

• バージョン管理

• より高度な機能

- テスト(Testing)
- デバッグ実行
- ベクトル・行列演算
- グラフ描画
- 後期
- (文字コード)
- (正規表現)
- (クラス)
- (オブジェクト指向)



<http://kray.jp/blog/git-pull-rebase/>

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？

1. プログラムの特徴
2. (プログラミングにおける2大原則)
3. (プログラミングを円滑に進めるための周辺技術)

2. 演習1:教科書のコードを動かしてみる

3. 演習2:オブジェクトと式と型

4. 演習3:変数と等号の利用、実行の様子

5. シラバス

6. 授業方針

7. 宿題・補足

実現したいことを理解し、手順として整理し、プログラミング言語で記述(翻訳)すること。ペアプロで互いに教え合おう。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

演習1:

教科書のサンプルコードを動かしてみる

1. 「ターミナル」を起動。
 - Finder => アプリケーション/ユーティリティ/ターミナル.app
 - Dockへ登録するか、ショートカット登録しておくとう便利。
2. Pythonインタプリタの起動。
 - 「python」と入力。
3. インタプリタにコードを入力。
 - e.g., 最初のコード例, 2.1節

```
print('Yankees rule!')
```
 - 注意: Python 2.x と 3.x とで書式が異なることがあります。参考書等を購入する際にはPython3を選ぶ。
4. インタプリタを終了
 - 「exit()」もしくは「Ctrl+D」。
 - Controlキーを押しながらDを1回押して離す。

「今日の目標」
手順1~4を数回繰り返して、教科書のコード例を試せるようになる。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1: 教科書のコードを動かしてみる
3. 演習2: オブジェクトと式と型
4. 演習3: 変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

ターミナル+Pythonインタプリタを使った作業工程に慣れよう。Python2と3の違いに注意。

演習2:

オブジェクトと式と型 (教科書2.1.1節)

- **オブジェクト (object)**
 - 型を持つ操作単位。
- **型 (type)**
 - 数字や文字等の種別のこと。
 - 整数: integer, int
 - 浮動小数点数 (小数): floating point numbers, float
 - 論理値: boolean value
 - True, False
 - 文字列: string, str
 - 空 (値を持たない状態)
 - None
- **式 (expression)**
 - オブジェクトと命令を紐付ける命令文。

```
oct:tnal% python
>>> 3 + 2
5
>>> 3.0 + 2.0
5.0
>>> 3 != 2
True
>>> type(3)
<class 'int'>
>>> type(3.0)
<class 'float'>
>>>
```

type()は、値の型を確認するための関数。
関数とは、ある特定の機能を提供する部品 (と当面は考えよう)。

プロンプト (>>>)とは、シェルやインタプリタが「ユーザからの入力を受付可能である」ことを明示するもの。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1: 教科書のコードを動かしてみる
3. **演習2: オブジェクトと式と型**
4. 演習3: 変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

代表的な型 (int, float, str)、型の確認方法と演算子を覚えよう。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

演習3:

変数の利用(教科書2.1.2節)

変数(variable)とは、計算結果等を一時的に保存する(後で利用する)ための名前付きの格納庫。

記号=(assignment)は、数学における等号(左右が等価)ではない。

コード,プログラム,スクリプト(教科書の例+α)

```
1 pi = 3
2 radius = 11
3 print(pi)
4 sent = 'radiusの値は11です'
5 print(sent)
6 area = pi * (radius**2)
7 print(area)
8 radius = 14
9 print(radius)
```

記号**は、N乗演算子。
記号()は、演算順序の優先順位を明示するための記号。

コードの意味

1. 変数piを用意し、右辺の評価結果(int型の値3)に紐付けろ≒保存しろ(bind, assign)。
2. 変数radiusを用意し、右辺の評価結果(11)を保存しろ。
3. 変数piの値を出力しろ。
4. 変数sentを用意し、右辺の評価結果(str型の文字列)を保存しろ。
5. 変数sentを出力しろ。
6. 変数areaを用意し、右辺の評価結果を割り当てろ。右辺には演算が指示されているので、その指示に基づいて評価せよ。
7. 変数areaの値を出力しろ。
8. 変数radiusを用意しようとしたが、radiusは既に存在しているので、中身を14に上書き保存しろ。
9. 変数radiusの値を出力しろ。

実行の様子: 1行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	
4番地	
5番地	
6番地	
7番地	
8番地	
9番地	



実行の様子: 2行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	
6番地	
7番地	
8番地	
9番地	

実行の様子: 3行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

変化なし

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	
6番地	
7番地	
8番地	
9番地	

実行の様子: 4行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	str型の「radiusの値は11です」
6番地	変数sent
7番地	
8番地	
9番地	

実行の様子: 5行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

変化なし

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	str型の「radiusの値は11です」
6番地	変数sent
7番地	
8番地	
9番地	

実行の様子: 6行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	str型の「radiusの値は11です」
6番地	変数sent
7番地	int型の「363」
8番地	変数area
9番地	

実行の様子: 7行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

変化なし

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	str型の「radiusの値は11です」
6番地	変数sent
7番地	int型の「363」
8番地	変数area
9番地	

実行の様子: 8行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	str型の「radiusの値は11です」
6番地	変数sent
7番地	int型の「363」
8番地	変数area
9番地	int型の「14」

Radiusの参照先
が変わった。

実行の様子: 9行目

コード(教科書の例+α)

```
pi = 3
radius = 11
print(pi)
sent = 'radiusの値は11です'
print(sent)
area = pi * (radius**2)
print(area)
radius = 14
print(radius)
```

実行後に残るデータ

変化なし

番地	内容
1番地	int型の「3」
2番地	変数pi
3番地	int型の「11」
4番地	変数radius
5番地	str型の「radiusの値は11です」
6番地	変数sent
7番地	int型の「363」
8番地	変数area
9番地	int型の「14」

覚えてほしいこと

- 等号(=)の意味

- 数学では、等式。「左辺と右辺は等価である」という意味。
- Python(プログラム)では、assignment(割り当て)やbinding(紐付け)のことを指し、一時的に名前の付いた保管庫へ保存する動作のこと。
- 次のように動作する。
 - (1)右辺をevaluate(評価≡実行)し、
 - (2)その結果を左辺の変数に保存する。

- 実行後に残るデータ

- 何をどう処理したかは覚えていない。
- コードを実行することで「得られた値」が一時的に残る。
- 等号を使って変数に紐付けることで、「得られた値」を再利用することができる。

- 逐次処理

- 順番通りに実行する。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

プログラミングにおける等号は、(1)右辺を評価(実行)して、(2)結果を変数に保存する。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

達成目標、評価方法等について必要な時に参照。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

予習・復習を前提に進める

- 予習・復習:教科書は各自の自習教材
 - 教科書読んで分からないことは自分から相談しよう。教科書を読んで分かることを、わざわざ授業時間に一緒にやるのは時間が勿体無い。
 - 授業では Part 1 (Chapter 7まで)をメインに扱う。約100ページ。余裕がある人は11章までやってから、Chapter 12以降を好きな順番でトライ！
 - 該当Chapterを4回は読もう。同じペースで読む必要はない。分かる部分はショートカットし、分からない部分を減らしていこう。
- オリジナル課題のすゝめ
 - 予習・復習とは別に、自身で取り組みたいことをやる時間も取れると良い。例えば、「2単位授業の自習4時間」のうち、予習・復習を平均して2~3時間で終え、残り時間を自身や仲間らとのプロジェクトに割り当てる等。

一人ではやれないことをサポート

- 授業でやること
 - 環境構築。
 - 重要な点に関する解説。デモ・演習。
 - 一人ではやれないこと(ペア・プログラミング)。
 - 思考や疑問を言語化する練習。
 - 伝えることを通して「手順の考え方」や「翻訳の仕方」の例に触れる機会を増やす。
 - 講義「プログラミング演習1」との連携。
 - 課題サポート等

教育目標

- できるとは、「次の一歩」が分かること
- 授業方針
 - しばらくは「例示」。
 - 「次の一歩」が分かるまで手を貸す。
 - E.g., 2年次以降ではスマートフォンのアプリ開発したりしますが、そこで使う言語「Swift」等、新しい言語は独学になります。
 - 独学できる力、調べ方、トラブルシューティングの力、、、(＝「次の一歩」を検討する力)を学ぼう。正解があるとは限らない状況において、「次の一歩」を検討し、行動に移せるようになろう。

プログラミングを勉強する際のポイント

- 疑問を持とう
 - この単語はどういう意味？
 - このコードは何をやろうとしているのだろうか？
 - 何故こう書くのだろうか？
- やりたいことを文章や口頭で説明できるまで理解しよう
 - 説明できないことを「コードで書く」のは無理。
- 「こうなるだろう」と想像して実行しよう
 - 想像と異なる動作をした場合、それは**学びを得るチャンス！** 要因を調査しよう。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

疑問に感じた点は
どんどん質問しよう。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

宿題

- 復習: 教科書読み
 - 1章 * 概要掴むぐらい(30分程度)でok
 - 2章~2.1.2節まで * 1~2時間程度想定
 - 2.1.3節はスキップ。
 - 余裕ある人は2.2節。
- 課題レポート: 授業ページ参照
- 自習(オススメ)
 - paiza
 - Python入門編1:プログラミングを学ぶ (全9回)
 - <https://paiza.jp/works/python3/primer>
 - progate
 - Python I, II
 - <https://prog-8.com>

補足

• 辞書アプリ

- アプリケーション->**辞書.app**
- (1) 起動して、左上の「辞書」メニューから「環境設定」を選び、
- (2) New Oxford American Dictionary、Oxford American Writer's Thesaurus あたりの**英英辞書**にチェック

和訳して読むというより、英語で読む練習をした方が有益

• 予習

- **内容を100%理解することは求めています。**分かりにくい部分に目星をつけて授業に参加するだけでも、理解度は大きく異なります。

• ミニテストの趣旨

- 復習内容についてはできるだけ正しく答えて欲しい。
- 予習内容については今は間違ってもOK。その日の授業で理解して欲しい。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？
 1. プログラムの特徴
 2. (プログラミングにおける2大原則)
 3. (プログラミングを円滑に進めるための周辺技術)
2. 演習1:教科書のコードを動かしてみる
3. 演習2:オブジェクトと式と型
4. 演習3:変数と等号の利用、実行の様子
5. シラバス
6. 授業方針
7. 宿題・補足

教科書・参考サイト参照しつつ、手を動かそう。

プログラミング1

(第1回) 卓上プログラミングによる開発設計概観、Pythonインタプリタの起動と逐次処理・変数の利用

1. プログラミングとは何か？

1. プログラムの特徴

2. (プログラミングにおける2大原則)

3. (プログラミングを円滑に進めるための周辺技術)

2. 演習1: 教科書のコードを動かしてみる

3. 演習2: オブジェクトと式と型

4. 演習3: 変数と等号の利用、実行の様子

5. シラバス

6. 授業方針

7. 宿題・補足

実現したいことを理解し、手順として整理し、プログラミング言語で記述(翻訳)すること。ペアプロで互いに教え合おう。

ターミナル+Pythonインタプリタを使った作業工程に慣れよう。Python2と3の違いに注意。

代表的な型(int, float, str)、型の確認方法と演算子を覚えよう。

達成目標、評価方法等について必要な時に参照。

疑問に感じた点はどんどん質問しよう。

プログラミングにおける等号は、(1)右辺を評価(実行)して、(2)結果を変数に保存する。

教科書・参考サイト参照しつつ、手を動かそう。

参考文献

- 教科書: Introduction to Computation and Programming Using Python: With Application to Understanding Data
- 来たるべき、「みんな」のコードのために: 平成4年生まれがつくるプログラマーの学校,
<http://wired.jp/2016/03/04/kusano-teacher/>
- 20歳を過ぎてからプログラミングを学ぼうと決めた人たちへ, <http://www.slideshare.net/ShuUesugi/20-9290892>
- (paiza) Python入門編1:プログラミングを学ぶ (全9回), <https://paiza.jp/works/python3/primer>
- (progate) Python I, II: <https://prog-8.com>

時間があつた場合のおまけ

翻訳に至る3ステップの例

プログラミングに含まれる3ステップ


- (コンピュータはプログラミング言語しか知らないので、)プログラマは、実現したいことをプログラミング言語に翻訳する必要がある。
 1. 「実現したいこと」の理解が大前提。この理解が不十分の場合、何を翻訳したら良いかが分からない。
 2. 「理解したこと」を手順として説明できるレベルまで整理する。
 3. 最後に、プログラムへ「翻訳(記述)」する。
 - 初学者は、プログラミング言語の構文・仕様を学ぶ必要がある。

一般的に、「プログラム」には3番目の翻訳結果しか残らない。1の理解や、2の整理が欠落してしまう傾向にある。レポートを書く時、また友人らと一緒に課題に取り組む際には、1,2の説明をするように心がけよう。

翻訳に至る3ステップの例 1/4

時間があれば最後に。

- Step 1: 実現したいこと
 - 受講生の中から一名をランダムに選ぶ。
- Step 2: 整理した手順
- Step 3: 翻訳




実現するためには、何をどの順番で実行したら良い？

翻訳に至る3ステップの例 2/4

- Step 1: 実現したいこと
 - 受講生の中から一名をランダムに選ぶ。
- Step 2: 整理した手順
 - 1. 受講生一覧を用意。
 - 2. 一覧をシャッフルする。
 - 3. シャッフルされた一覧から、頭の一人を抜き出す。
- Step 3: 翻訳

翻訳に至る3ステップの例 3/4

- Step 1: 実現したいこと
 - 受講生の中から一名をランダムに選ぶ。
- Step 2: 整理した手順
 - 1. 受講生一覧を用意。
 - 1-1. 受講生は学籍番号と一対一対応している。
 - 1-2. 学籍番号は195701～195767。
 - 2. 一覧をシャッフルする。
 - 3. シャッフルされた一覧から、頭の一人を抜き出す。
- Step 3: 翻訳

翻訳難しそうなら、より細かい手順を考えてみる。

翻訳に至る3ステップの例 4/4

- Step 1: 実現したいこと
 - 受講生の中から一名をランダムに選ぶ。
- Step 2: 整理した手順
 - 1. 受講生一覧を用意。
 - 受講生は学籍番号と一対一対応している。
 - 学籍番号は195701～195765。
 - 2. 一覧をシャッフルする。
 - 3. シャッフルされた一覧から、頭の一人を抜き出す。
- Step 3: 翻訳

- Step 3: 翻訳
 - # 1. 受講生一覧を用意。
`students = list(range(195701, 195767))`
 - # 2. 一覧をシャッフル。
`import random`
`random.shuffle(students)`
 - # 3. 頭一人を抜き出す。
`students.pop()`

手順の考え方と翻訳技術を学ぶ

代表的な原則

- **Done is better than perfect.**
 - まずは動くものを。
- **KISS原則, DRY原則**
 - Keep it simple, stupid!
 - 小さく作る。(小さく分解して、組み合わせる)
 - Don't repeat yourself.
 - 繰り返しを避ける。

経験を積む

- 当面は決められた仕様を「**どう実装するか**」。
 - どう機能分解するか？
 - どんな手順になるか？
 - どう構造化するか？
- 少しずつ検討範囲拡大。
- **ペア・プログラミング**。
 - 互いに教え合う、言語化練習
 - 知識の共有化