# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 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
  - コンピュータ
  - 命令
  - コンサートとかパンフレットに載ってる
  - パソコン
  - アプリケーション

# 「プログラム」という言葉から連想するものは？

- 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。
- クイズ2：他と比べて上記(3)が持つ側面・特徴は？

# (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。
-> 特徴は？

- 授業メモ
  - 他のは、予め作られてるが、コンピュータの方は自分で作っていきそう
  - パソコンを使うか使わないか

# (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。
# -> 特徴は？

- 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,,,

- 複製が容易
  - コストはほぼ無視できる
- 再現が容易
  - 書いた通りに動く
- （条件付きで）編集や再利用が容易
  - テスト（Testing）
  - ユニット・テスト
  - バージョン管理

# プログラミングとは？

- 広義（広辞苑）
  - 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―が気に入らない」
  - (2) →仕様書に同じ。

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# プログラムの中心: 関数 (Functions)

http://www.slideshare.net/ShuUesugi/20-9290892

- 全てのプログラムはたくさんの関数で構成される
  - 数学における関数の例
  - 1次関数、2次関数、三角関数、、、
  - 似ている点: 様々な関数を組み合わせることで、より複雑な関数を作ることができる

# 卓上プログラミングしてみよう！(1)

- Level 1: 縦5cm, 横6cmの四角形の面積を求めてください

# 卓上プログラミングしてみよう！(2)

- Level 1: 縦5cm, 横6cmの四角形の面積を求めてください
- Level 2: 縦5cm, 横4cmの四角形の面積を求めてください

# 卓上プログラミングしてみよう！(3)

- Level 1: 縦5cm, 横6cmの四角形の面積を求めてください
- Level 2: 縦5cm, 横4cmの四角形の面積を求めてください
  - tate = 5; yoko = 4
  - tate * yoko -> 20
  - menseki(tate, yoko) = tate * yoko
  - tate = 10; yoko = 8
  - menseki(tate, yoko) -> 80
  - output = menseki(tate, yoko)

# 卓上プログラミングしてみよう！(4)

- Level 1: 縦5cm, 横6cmの四角形の面積を求めてください
- Level 2: 縦5cm, 横4cmの四角形の面積を求めてください
- Level 3: 1辺の長さ5cmの立方体の体積を求めてください

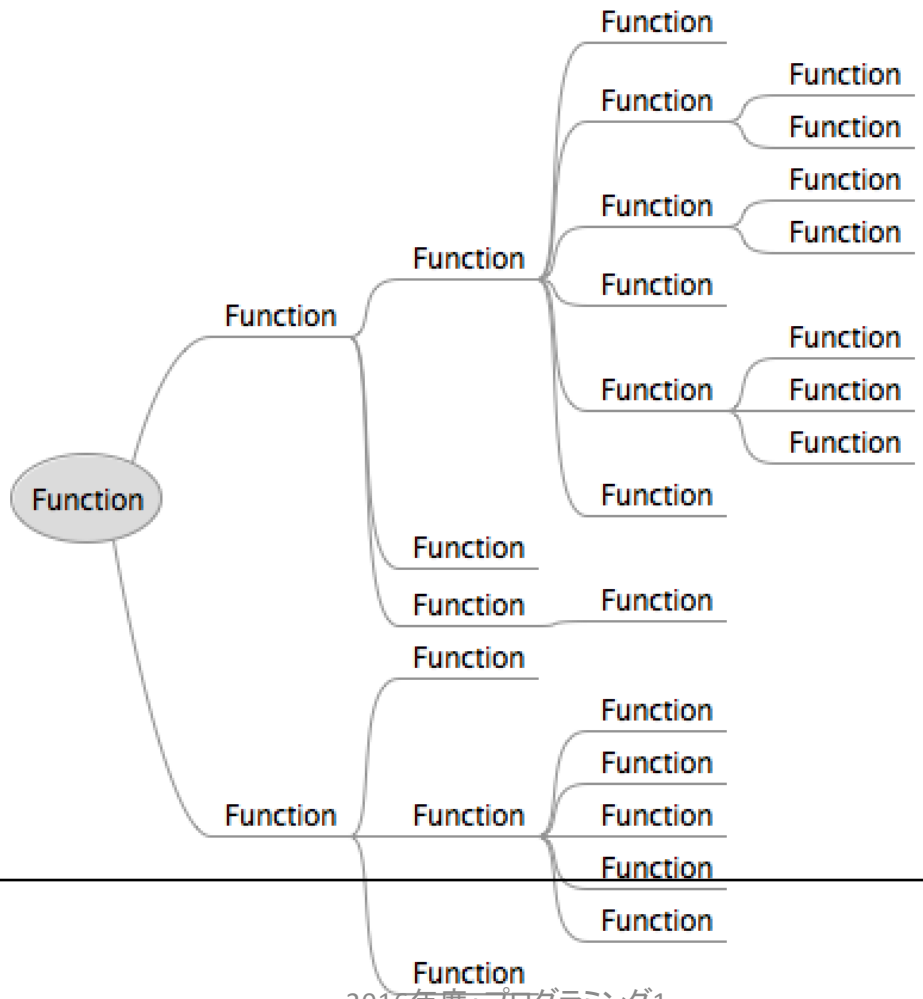
# 卓上プログラミングしてみよう！(5)

- Level 1: 縦5cm, 横6cmの四角形の面積を求めてください
- Level 2: 縦5cm, 横4cmの四角形の面積を求めてください
- Level 3: 1辺の長さ5cmの立方体の体積を求めてください
  - tate = 5, yoko = 5, takasa = 5
  - tate * yoko * takasa -> 125
  - taiseki(tate, yoko, takasa) = tate * yoko * takasa
  - taiseki(tate, yoko, takasa) = menseki(tate, yoko) * takasa

# プログラムの中心 : 関数 (Functions)

http://www.slideshare.net/ShuUesugi/20-9290892

- 全てのプログラムはたくさんの関数で構成される



# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# プログラミングの一部: 仕様 (Design / Specification)

- 仕様を決定するのは難しい

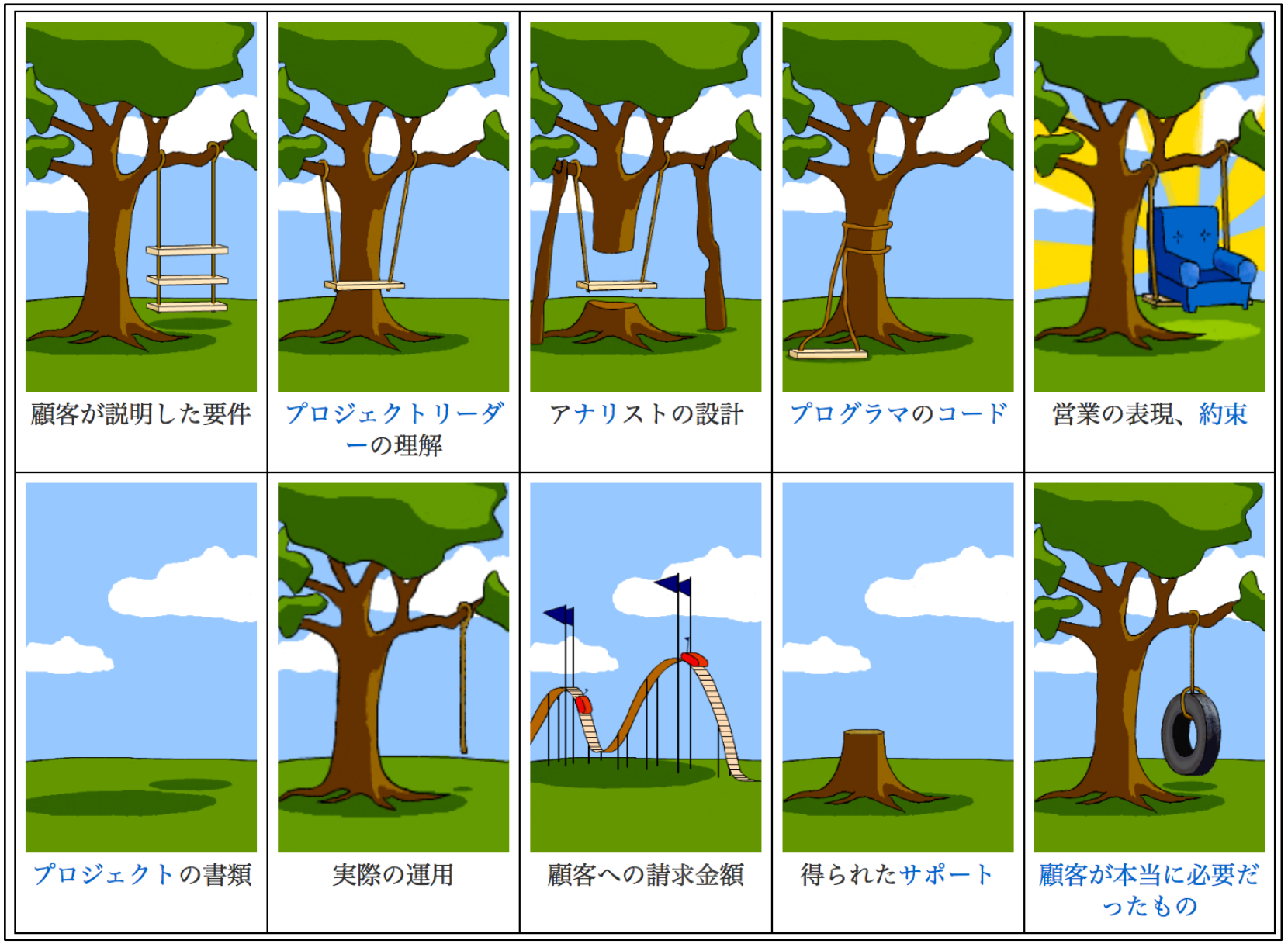
# 卓上プログラミング2:
# 室温を25度に保つ空調をプログラムしてみよう！

- 授業メモ
  - 暑い時に空調をつけると温度を下げてくれる
  - 25度という基準を作り、超えたら空調をつける。以下だったら消す。

# 卓上プログラミング2:
# 室温を25度に保つ空調をプログラムしてみよう！

- 「もし『25℃に室温を保つ空調のプログラム』を依頼されたら、どうすればいいか」
- 草野翔に、プログラミング教育についてたずねると、彼はまずそんな思考実験をはじめた。「25℃」という温度が正確なのか、1年間で外気はどのように変動するのか、もし障害が発生したときには何分以内に25℃に戻さないといけないのか。そして、何のために「25℃に保つ」必要があるのか。つまる所、それらの条件を確かめるために依頼主に会いに行く必要があるという。それがわからないままコードを書きはじめるプログラマーは、草野の基準によれば、優秀なプログラマーではない。
- 出典: http://wired.jp/2016/03/04/kusano-teacher/

# 仕様決定には様々な要因が絡む




http://dic.nicovideo.jp/a/顧客が本当に必要だったもの

# 関数や仕様はどう決定したら良いか？

授業計画:
第1回～第8回

- 代表的な原則
- KISS原則
  - Keep it simple, stupid!
  - 小さく作り、組み合わせる。
    - 一つの関数は一つの作業をこなす。
  - 各部品(関数)をテストする。
    - 検証・再現性を意識する。
- DRY原則
  - Don’t repeat yourself.
  - 繰り返しを避ける。

授業計画:
第2回～

- 経験を積む
  - 様々な問題にトライする。
  - ペア・プログラミング。
    - 互いに教え合う
    - 知識の共有化
- (マルチステークホルダーの存在を意識する)

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 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，関数と再利用

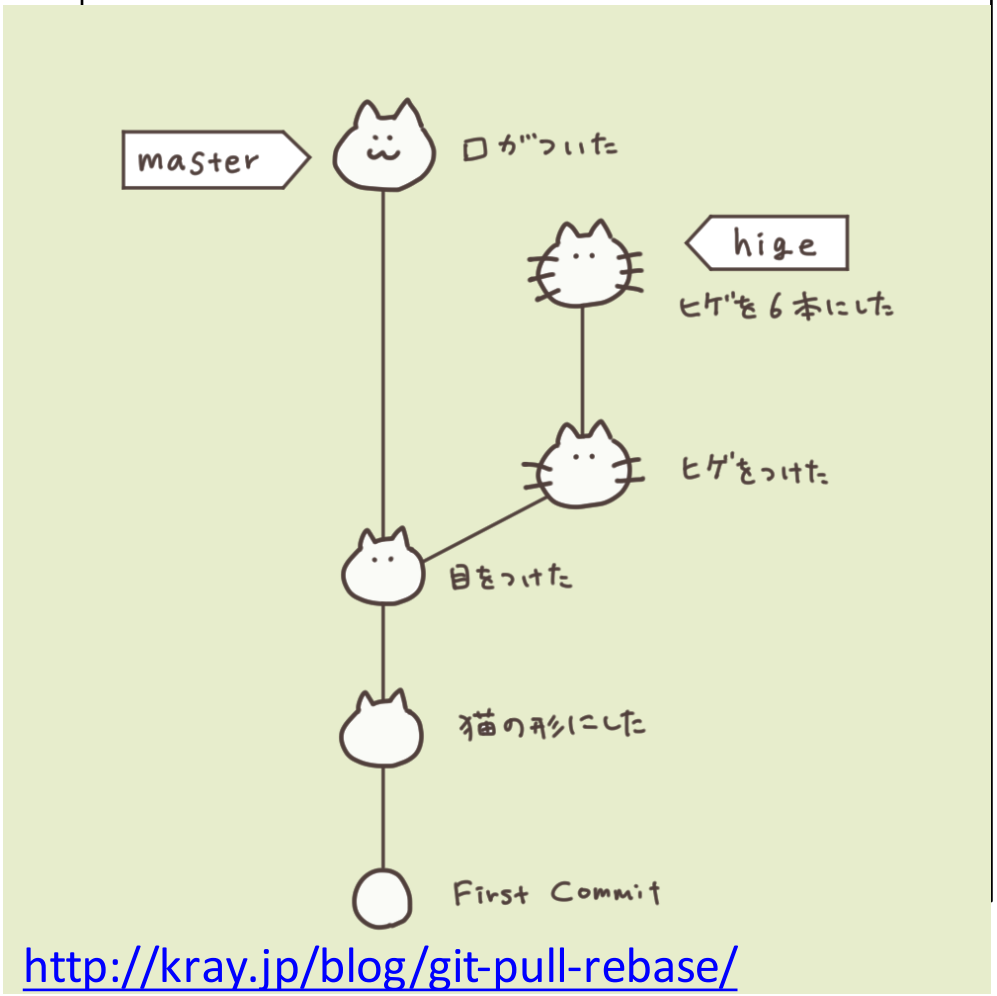
1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# プログラミングを円滑に進めるために

授業計画:
第9回～

- バージョン管理




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

授業計画:
第10回～

- より高度な機能
  - ベクトル・行列演算
  - グラフ描画
  - 文字コード
  - 正規表現
  - クラス
  - オブジェクト指向
- 様々な演習や課題

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# シラバス

- https://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/syllabus.html

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 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 8まで)をメインに扱う。約100ページ。これを4回は読もう。余裕がある人はChapter 9以降を好きな順番でトライ！
  - 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。（課題設定時には、平均2,3時間で終えられることを想定した分量を意識するようにします）

# 一人ではやれないことをサポート

- 授業でやること
  - 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)。
    - ペアプログラミングのやりかた: http://goo.gl/ZWtlW
  - 講義「ソフトウェア演習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
  - 「次の一歩」が分かるまで手を貸す

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 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，関数と再利用

1. プログラミングとは何か？
    1. 全てのプログラムはたくさんの関数で構成される
    2. 仕様を決定するのは難しい
        1. 代表的な原則：KISS原則、DRY原則
        2. 知識の共有化、互いに教え合う：ペア・プログラミング
    3. 円滑に進めるために
        1. バージョン管理
        2. より高度な機能
2. シラバス
3. 授業方針
    1. 予習・復習を前提に進める
    2. 一人ではやれないことをサポート
    3. 「次の一歩」が分かるまで手を貸す
4. 宿題
5. サンプルコードの動かし方

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

# 宿題

- 予習1: 教科書読み
  - 1章
  - 2章～2.1.3節まで
- 予習2(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
    - やれる範囲でok

# サンプルコードの動かし方

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便利。
2. インタプリタの起動。
   - 「python3」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, p.9, 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。教科書の脚注参照。
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を押す

# 参考文献


- 教科書: Introduction to Computation and Programming Using Python, Revised And Expanded Edition
- UNIXという考え方
- 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/
- 顧客が本当に必要だったもの, http://dic.nicovideo.jp/a/顧客が本当に必要だったもの
- 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
- git pull と git pull –rebase の違いって？図を交えて説明します！, http://kray.jp/blog/git-pull-rebase/
- ペアプログラミングのやりかた, http://goo.gl/ZWtlW
- (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