復習

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：変数と等号の利用、実行の様子
4. シラバス
5. 授業方針
6. 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。KISS原則、DRY原則を意識しよう。ペアプロで互いに教え合おう。テスト・バージョン管理・デバッグ実行で円滑に開発を進めよう。

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。

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)結果を変数に紐付ける。

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。

教科書・参考サイト参照しつつ、手を動かそう。

講義ページ: http://ie.u-ryukyu.

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

教科書に書かれていないか、後回しになってる内容

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

Chapter 1 の補足1

複数回に分けて補足します

Chapter 1,
A Computer does two things

- Calculations (計算)
- Remembers the results of those calculations (計算結果を覚える)

Chapter 1, Computational thinking (計算的思考)

Declarative knowledge (宣言的知識)

- composed of statements of fact. (明確な事実の宣言)
- "the square root of x is a number y such that y*y = x."

Imperative knowledge (命令的知識)

- "how to" knowledge, or recipes for deducing information. (情報を導くためのレシピ)
- start with a guess, g.
- if g*g is close enough to x, stop and say that g is the answer.
- otherwise create a new guess by averaging and x/g, i.e., (g+x/g)/2.
- Using this new guess, which we again call g, repeat the process until g*g is close enough to x.

Algorithm (アルゴリズム)

An algorithm is a finite list of instructions that describe a computation that when executed on a provided set of inputs will proceed though a set of well-defined states and eventually produce an output.

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

欲しい出力を得るためのレシピを考える必要がある。
レシピ≒アルゴリズム。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

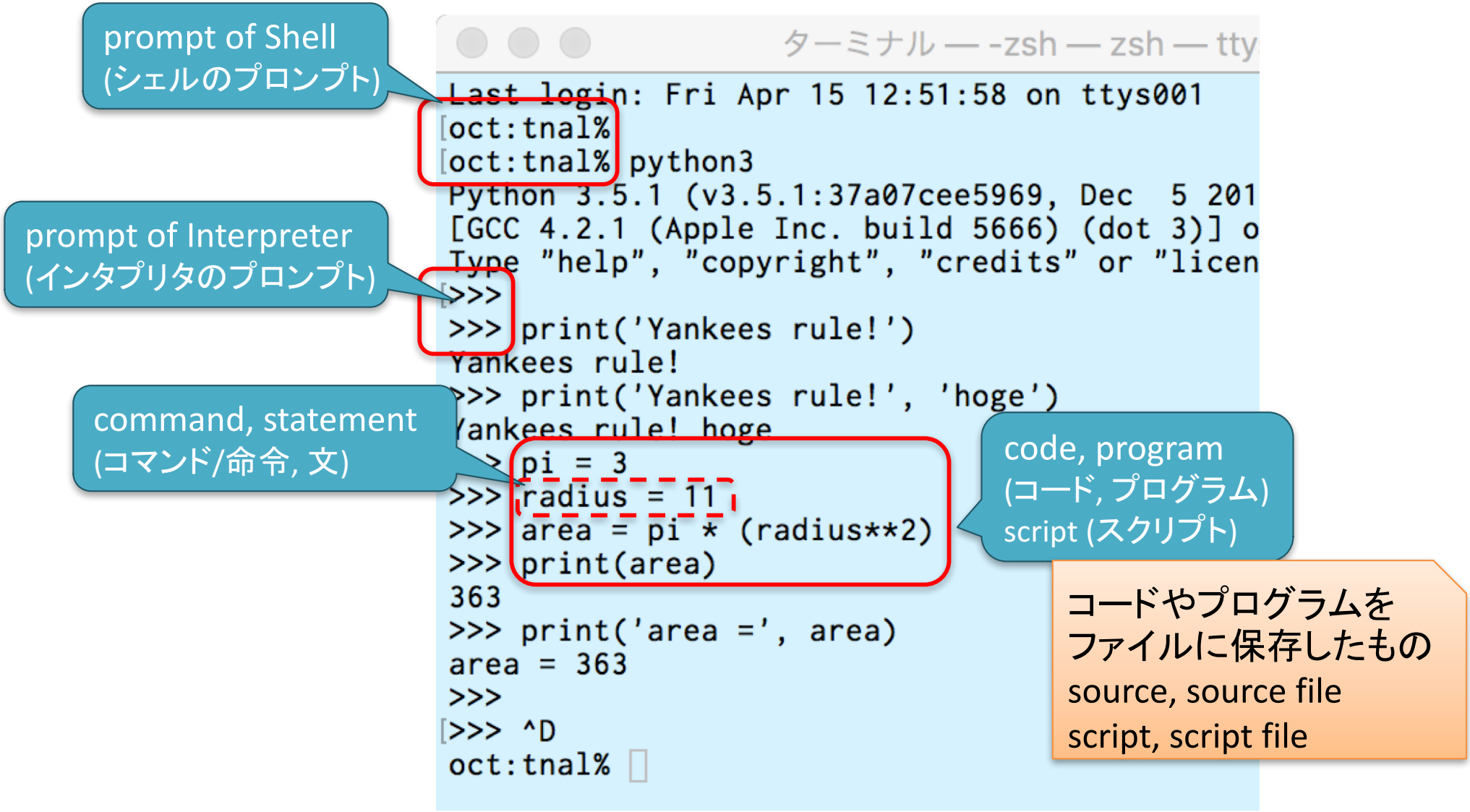
Chapter 2 -- 2.1.2までの補足

Glossaries, 用語集1, 2

[教科書] 2.1.2 変数と代入

Reserved words, 予約語

Glossaries, 用語集 1



Glossaries, 用語集 2

operator
(オペレータ, 演算子)

type of object
(オブジェクトの型)

variable
(経数)

assignment
(代入文)

```
>>> 3 + 2
5
>>> type(5)
<class 'int'>
>>> a = 3 + 2
>>> type(a)
<class 'int'>
>>> type(5.0)
<class 'float'>
>>>
>>> 3 + 2 == 5
True
>>> 3 + 2 == 4
False
>>> 3 + 2 != 4
True
>>> type(True)
<class 'bool'>
>>> type(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>> type('true')
<class 'str'>
>>>
```

int: integer (整数)
float: floating point number (浮動小数点数)
str: string (文字列)
bool: boolean values, True/False (ブール型)
None: lack of value, (空のデータ)
See also,
https://docs.python.org/3.6/library/stdtypes.html#built-in-types

comparison (relational) operator
(比較演算子)

a **and** b: a, b共にTrueの場合にTrueを返す
a **or** b: a, bどちらかがTrueの場合にTrueを返す
not a: aがFalseの場合にTrueを返す

大文字小文字で意味が異なる。
シングルクォート(')の有無でも変わる。
Trueは予約語のboolean value。
trueはここでは未定義の変数。
'true'はstring。

[教科書] 2.1.2 変数と代入

- 「=」は変数とオブジェクトを紐付ける。
  - 「=」の右辺を評価(evaluate)し、その結果を左辺に代入(assignment)する。
  - コード例

```
1 pi = 3
2 radius = 11
3 area = pi * (radius **2)
4 print(area) #-> 363
5 radius = 14
6 print(area) #-> 363
7 area = pi * (radius **2)
8 print(area) #-> 588
```

変数areaは右辺の評価結果を保存する

radiusを変更しても、変数areaには影響がない

変数areaは7行目の「=」により、右辺の評価結果が保存される

Reserved words, 予約語

https://goo.gl/4TclUz

- 一覧（赤丸は今回出てきた予約語）

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

レシピを記述するための道具（基本的な型・算術演算子・比較演算子・論理演算子・変数・代入文）を覚えよう。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

数値演算と文字列演算の補足とprint()の応用編

（教科書に書かれてません）

数値演算と文字列演算

```
oct:tnal% python3
Python 3.6.0 (defau
[GCC 4.2.1 Compatib
Type "help", "copy
>>> # 数値の演算
... 3 + 11
14
>>> pi = 3
>>> pi + 11
14
>>>
>>> # 文字列の演算
... 'スライム' + 1
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: must be str, not int
>>> 'スライム' + 'が2体現れた'
'スライムが2体現れた'
>>> enemy = 'スライム'
>>> enemy + 'が2体現れた'
'スライムが2体現れた'
>>> output = enemy + 'が2体現れた'
>>> print(output)
スライムが2体現れた
>>>
```

comment (コメント)
何も実行しない

数値と文字列を直接足すことはできない
TypeError: must be str, not int
型エラー: str型であるべき

文字列同士の足し算は、結合になる。

print()の応用編

```
>>> enemy = 'スライム'
>>>
>>> # case 1: 出力したい文字列を1変数に用意してからprint()する。
... output = enemy + 'が2体現れた'
>>> print(output)
スライムが2体現れた
>>>
>>> # case 2: print()内で演算処理する。
... print(enemy + 'が2体現れた')
スライムが2体現れた
>>>
>>> # case 3: str.format()形式（Python公式）を利用する。
... print('{0}が2体現れた'.format(enemy))
スライムが2体現れた
>>>
```

str.format()形式
「'文字列'.format(変数)」
文字列中の{0}等を変数に置き換えて文字列を作成。

```
>>> enemy = 'スライム'
>>> num = 3
>>> '{0}が{1}体現れた'.format(enemy,num)
'スライムが3体現れた'
>>> output = '{0}が{1}体現れた'.format(enemy,num)
>>> print(output)
スライムが3体現れた
>>>
```

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

基本演算とprint()書式を覚えよう。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

スクリプトの利用
（教科書に書かれてません）

スクリプトとは？
スクリプトを書いて動かしてみよう
スクリプト vs.インタプリタ

スクリプトとは？

- これまで
  - インタプリタ上で直接コードを入力して実行する。
  - 利点: 入力したコードの結果をその場で見れる。
  - 欠点: 同じコードを複数回実行したい場合には、その都度タイプする必要がある。
- 別の方法
  - コードをファイルに保存（拡張子は「.py」）。
  - コードが保存されたファイルを「source file, source code, script file, script code」等と呼ぶ。
  - 保存したファイル名が「test.py」ならば、ターミナル上で次のようにコマンドを実行すると、スクリプトを実行できる。

```
% python3 test.py
```

冒頭の「%」は、シェルのプロンプトを表しています。実際にタイプするコマンドは「python3 test.py」

スクリプトを書いて動かしてみよう

- やってみよう
  - インタプリタ上で「print('hello!')」と入力してエンターキーを押すと、「hello!」と返してくる。
  - これをスクリプトファイル「test.py」として保存するには、エディタ（emacs）コマンドを、ファイル名を指定して起動する。

```
% emacs test.py
```

  - emacsでファイルtest.pyに「print('hello!')」と書いて、保存(Ctrl-x, Ctrl-s)する。
  - 保存し終えたら、エディタを終了(Ctrl-x, Ctrl-c)する。
  - シェル上で次のようにスクリプトを実行する。

```
% python3 test.py
```

スクリプトファイル vs.インタプリタ

- ポイント
  - コードをスクリプト(ファイル)に保存する。
  - ファイルに保存するにはエディタを使う。
  - スクリプトを実行するには「python3 ファイル名」として、シェル上で実行する。
  - 作業中は「ターミナル」「エディタ」「インタプリタ」を切り替えながら作業することになる。(複数ターミナル立ち上げるのも手)
- 利点
  - 一度動くように仕上げたら、後は中身を見なくても同じ動作を再現することができる。
- 欠点
  - 初学者だと、1行ずつコードの実行結果を確認できるインタプリタが便利。
- オススメ
  - インタプリタで大まかな流れを確認し、ある程度コード全体の見積もりができたらスクリプトファイルとして整理する。

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

インタプリタ実行とファイル実行を使い分けよう。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

変数名・ファイル名の命名規則
（教科書に書かれてません）

変数名・ファイル名の命名規則

- 変数名やファイル名に使える文字
  - 原則として「英数字」と「_ (underscore)」。
  - 大文字小文字は区別される。
  - 冒頭に数字は使えない。
- 変数名・ファイル名を適切に選択する
  - 「適切」？
    - Level 1: その変数が表す用語の英単語（小文字）を使う。
    - Level 2: 複数単語で命名したいなら「_ (underscore)」で繋げて書く。
    - Level 3: 同じモジュール・クラス内では統一規約を採用する。
    - Level 4: 英単語の微妙なニュアンスの差に気をつける。
  - 規約の例: Google Python Style Guide
    - https://google.github.io/styleguide/pyguide.html
    - Naming
      - Modules（ファイル）: lower_with_under
      - Local Variables: lower_with_under

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

慣習を守ることで「他人が読みやすいコード（readable code）」になる。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

マニュアルの参照
（教科書に書かれてない
or
後回しになってます）

マニュアル

- 公式ドキュメント
  - https://docs.python.org/3.6/index.html
- help()コマンド
  - help(print)
- Google先生
  - e.g.,「python3 print」
  - 注意
    - Python 2 <-> Python 3で異なることがある。
    - Web上の情報は間違ってることがある。

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

help()やオンラインマニュアルを活用しよう。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

演習

初めてのレポート
ペア・プログラミング

ペアプロ演習

- 授業ページを参照

宿題

- 復習: 適宜
- 課題レポート1: 〆切: 第3回(4/27)の講義開始前
- 予習1: 教科書読み
  - 2章
    - (スキップ) 2.1.3
    - 2.2 Branching Programs
    - 2.3 Strings and Input
    - 2.4 Iteration
  - (スキップ) 3章
  - 4章
    - (スキップ)4章冒頭
    - 4.1.1 Function Definitions
- 復習・予習(オススメ): paiza, progate

プログラミング1

(第2回) Pythonインタプリタとスクリプトの体験1，ペア・プログラミングの導入

1. Chapter 1 の補足1
    1. Calculations and Remembers
    2. Computational thinking
2. Chapter 2 -- 2.1.2までの補足
    1. Glossaries, 用語集1, 2
    2. [教科書] 2.1.2 変数と代入
    3. Reserved words, 予約語
3. 数値演算と文字列演算の補足とprint()の応用編
4. スクリプトの利用
    1. スクリプトとは？
    2. スクリプトを書いて動かしてみよう
    3. スクリプト vs.インタプリタ
5. 変数名・ファイル名の命名規則
6. マニュアルの参照
7. 演習
8. 宿題

欲しい出力を得るためのレシピを考える必要がある。
レシピ≒アルゴリズム。

レシピを記述するための道具（基本的な型・算術演算子・比較演算子・論理演算子・変数・代入文）を覚えよう。

基本演算とprint()書式を覚えよう。

インタプリタ実行とファイル実行を使い分けよう。

慣習を守ることで「他人が読みやすいコード（readable code）」になる。

help()やオンラインマニュアルを活用しよう。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/

参考文献

- 教科書: Introduction to Computation and Programming Using Python, Revised And Expanded Edition
- Python 3.6.1 documentation, https://docs.python.org/3.6/index.html
- Google Python Style Guide, https://google.github.io/styleguide/pyguide.html