復習

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, 3
    2. Reserved words, 予約語
3. 文字列結合の例
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）を使えるようになろう。

基本演算とstr.format書式を読めるようになろう。

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/2019/prog1/

プログラミング1

(第3回) インタプリタとスクリプトの体験2: 文字列とif文, 関数の利用

1. Chapter 2.2, 3.1, 4.1.1の補足
    - 2.2 Branching Programs (条件分岐)
    - 3.1 Looping (繰り返し処理)
    - 4.1.1 Function Definitions (関数定義)
    - Reserved words, 予約語
2. ペアプロ演習
3. 宿題

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2018/prog1/

Branching Programs (条件分岐)とは？

- これまでのプログラム
  - インタプリタやファイルに書かれた命令文を、上から下に向かって順序よく実行する。
  - これだけだと、以下のような処理を書けない。
    - ゲームアプリで、レビュー書いたユーザに対して特別報酬を与えたい。
    - Webサービスで、アカウントを作成してログインしたユーザ向けに専用ページを表示したい。

- 条件分岐の考え方
  1. ある条件を満足しているか否かを確認する。(条件判定し、True/Falseいずれなのかを確認する。
  2. 判定結果に基づき、True時の処理(True block)、False時の処理(False block)を分けて記載する。True blockだけでもok。

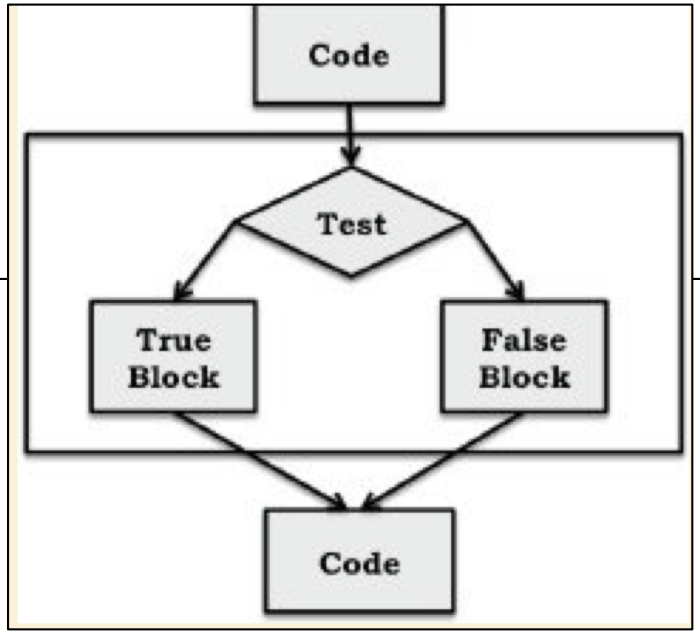
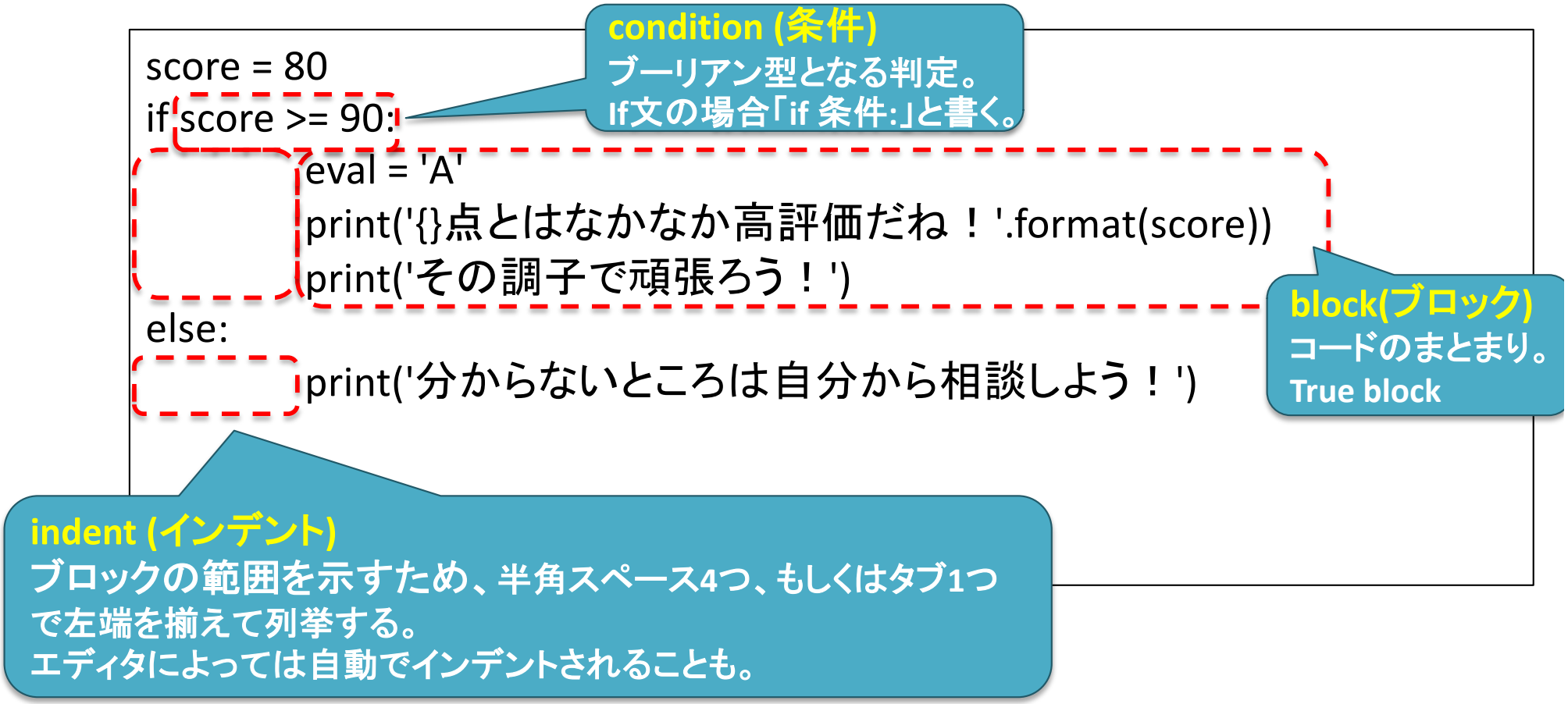


Fig. 2.3 Flow chart for conditional statement

条件分岐（if文, if-statement）の例1



条件分岐（if文, if-statement）の例2

```
score1 = 80
if score >= 90:
        eval = 'A'
        print('{}点とはなかなか高評価だね！'.format(score))
        print('その調子で頑張ろう！')
elif score >= 80:
        eval = 'B'
elif score >= 70:
        eval = 'C'
elif score >= 60:
        eval = 'D'
else:
        print('分からないところは自分から相談しよう！')
```

elif = else if。最初の条件「score>=90」に当てはまらない時（else時）に、改めて確認したい条件（if）があるという書き方。ここでは「80以上」という判定しか書いていないが、実際には「90以上ではない」ことが前提になったブロックになっている。

プログラミング1

(第3回) インタプリタとスクリプトの体験2: 文字列とif文, 関数の利用

if文を使い、条件分岐できるようになろう。ブロックを指定するためのインデントを忘れずに。

1. Chapter 2.2, 3.2, 4.1.1の補足
   - 2.2 Branching Programs (条件分岐)
   - 3.2 For loop (反復処理)
   - 4.1.1 Function Definitions (関数定義)
   - Reserved words, 予約語
2. ペアプロ演習
3. 宿題

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

looping (繰り返し処理)とは？

- これまでのプログラム
  - 逐次処理＋条件分岐。
  - これだけだと、以下のような処理を「書くのに苦労」する。
    - 同じ処理を100回繰り返す。

- 繰り返し処理の種類
  - for
    - 指定した回数繰り返す。
    - 集めた要素に対して繰り返す。
  - while
    - ある条件を満足している間、繰り返す。（満足しなくなったらやめる）

3.2節 For Loops (反復)
＊1つ目の反復制御

「range型オブジェクト」の中身を確認したいなら、list型にキャストしよう。

```
# 確認例
data = range(1, 4)
print(list(data))
```

range ()関数
range(stop): 0～stop-1までの全int型オブジェクトを生成
range(start, stop): start～stop-1

```
for i in range(4):
    print(i)
```

for文
「in ～」で指定されたシーケンス集合（連続したデータ）に対して、
(1)1つずつ要素を取り出し、
(2)その要素を対象としてブロックを実行（反復処理）。
(3)全要素に対して(2)を実行し終えたらfor文を終了。

シーケンス集合の例: str, range, list

反復処理の例2

List（リスト）
順番の付いた要素を1つのオブジェクトとしてまとめたもの。
どんな型でも順番付きで格納できる。

```
scores = [80, 60, 50, 'hogege']
for i in scores:
    print(i)
```

プログラミング1

(第3回) インタプリタとスクリプトの体験2: 文字列とif文, 関数の利用

1. Chapter 2.2, 3.2, 4.1.1の補足
   - 2.2 Branching Programs (条件分岐)
   - 3.2 For loop (反復処理)
   - 4.1.1 Function Definitions (関数定義)
   - Reserved words, 予約語
2. ペアプロ演習
3. 宿題

まずはシンプルな繰り返しに慣れよう。

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

Functions（関数）とは何か？

- 関数とは、レシピ・手順に名前をつけたもの。
- 料理なら、材料を用意して、レシピ通りに誰かが調理したら、料理が仕上がる。
- プログラムなら、必要な入力情報を用意し、関数を呼び出したら（＝コンピュータに実行させる）、想定した出力が得られる。

- 関数の書き方
  1. 実現したい事象について整理し、手順（レシピ）を検討する。
  2. そのレシピに名前をつける。（関数名）
  3. 必要な入力情報（inputs, arguments, 引数；ひきすう）を指定する。
  4. 手順をブロックとして記述する。
- 関数の呼び出し方（実行方法）
  - *関数名(引数)*

関数の例1（戻り値がないケース）

def funcion_name(parameters):
- def: 関数宣言
- funciton_name: 関数名
- parameters: パラメータ
  ＊argments(引数)とも呼ぶ。処理に必要なデータ。
  料理と異なり、材料が不要なら引数なしでもOK。

Tips: 関数名の命名規則
lower_with_under()

```
# 敵に遭遇した際のメッセージを出力する関数
def encount_enemy(name, number):
    print('{}{}匹に遭遇した。'.format(name,number))
    print('勇者は逃げ出した。')

encount_enemy('スライム', 2) #関数実行の例（必要な入力を与え、関数を呼び出す）
encount_enemy('ドラキー', 1)
```

indent

block(ブロック)
コードのまとまり。

関数の例2（戻り値があるケース）

```
# 第1引数と第2引数を比較し、大きい方を返す関数。
def max(value1, value2):
        if value1 > value2:
                return value1
        else:
                return value2

# 関数実行の例1。出力されない。
max(10, 5)

# 関数実行の例2。
# step 1:「=」を使い、右辺を評価。
# step 2: 関数呼び出しにより実行され、return文の結果が評価結果になる。
# step 3: 評価結果が、左辺の変数resultに紐付けられる。
result = max(10, 5)
print(result)
```

return文
(1) return文に辿り着いたら、その**関数の実行をここで終える**。（それ以降のコードは実行しない）
(2) 関数の呼び出し元にreturn指定した**オブジェクトを返す**。

関数定義から見たblock

関数を実行する際には、
関数名(引数)　or　関数名()
のように、丸括弧付きで記載。引数を必要としない関数であっても、丸括弧は必要。

コード例（まとめバージョン）

```
1   scores = [80, 60, 100, 0, 90]
2
3   def change_score_to_rank(score):
4       if score >= 90:
5           result = 'A'
6       elif score >= 80:
7           result = 'B'
8       elif score >= 70:
9           result = 'C'
10      elif score >= 60:
11          result = 'D'
12      else:
13          result = 'F'
14      return result
15
16  for i in scores:
17      rank = change_score_to_rank(i)
18      print('{}の評価は{}です'.format(i, rank))
```

プログラミング1

(第3回) インタプリタとスクリプトの体験2: 文字列とif文, 関数の利用

1. Chapter 2.2, 2.3, 4.1.1の補足
   - 2.2 Branching Programs (条件分岐)
   - 3.2 For loop (反復処理)
   - 4.1.1 Function Definitions (関数定義)
   - Reserved words, 予約語
2. ペアプロ演習
3. 宿題

オリジナルの**関数（≒レシピ）を定義**できるようになろう。

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

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	

演習

前回の続き: 初めてのペア・プログラミング

プログラミング1

(第3回) インタプリタとスクリプトの体験2: 文字列とif文, 関数の利用

1. Chapter 2.2, 2.3, 4.1.1の補足
   - 2.2 Branching Programs (条件分岐)
   - 3.2 For loop (反復処理)
   - 4.1.1 Function Definitions (関数定義)
   - Reserved words, 予約語
2. ペアプロ演習
3. 宿題

if文を使い、条件分岐できるようになろう。ブロックを指定するためのインデントを忘れずに。

まずはシンプルな繰り返しに慣れよう。

オリジナルの関数(≒レシピ)を定義できるようになろう。

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2019/prog1/

宿題

- 復習: 適宜(これまでの内容)
  - 課題レポート2　＊講義ページ参照。
- 予習: 教科書読み
  - 3章
    - 3.1 Exhaustive Enumeration
    - 3.2 For Loops
- 復習・予習(オススメ): paiza, progate

参考文献

- 教科書: Introduction to Computation and Programming Using Python, Revised And Expanded Edition
- Reserved words, https://goo.gl/4TclUz