演習や課題レポートへの補足

- 設問文の意図がわからない場合
  - どんどん聞いてください！
- 課題レポート、演習等について
  - 出力結果を読もう。
  - 「エラーが出たまま次に進んでいる」ケース
  - エラーに気づいて、その後やり直してる分にはOK。できれば、その過程（エラーをどう解釈したのか、それを踏まえて何故そうやり直そうとしたのか）を言語化して欲しい。
  - 学籍番号、氏名を書こう。

- ターミナルのコピペ
  - できるだけ「背景色と文字色」が同系色にならないようにして欲しい。
  - e.g.,「明るい背景色＋明るい文字色」は読みづらい。
- 変数名
  - 命名規則を参考に、適切な変数名を付けよう。
    - なるべく「a」とか付けない。変数名から推測しやすくなるように名付けてみよう。

復習

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/2020/prog1/

プログラミング1

(第5回) スコープ、while文とループ処理の補足

1. Chapter 4.1.1の補足2
    1. 関数とローカル変数
2. Chapter 3.1の補足
    1. Iteration, looping (反復処理)
    2. ループ処理の例、実行例
    3. 3種類の処理流れ制御
    4. ループ文の補足
3. 演習
4. 宿題

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2020/prog1/

Chapter 4.1.1, 3.1の補足

4.1.1 Function Definitions

3.1

4.1.1 Function Definitions (関数定義)

・一連の手続きにmuliplyという名前を付けた。
・multiplyという関数は2つの引数を必要とする。
・x, y, a, b は関数内での呼び名（**ローカル変数**）。
・変数名は命名規則を意識して命名しよう。（下記は悪い例）

関数外のことは気にせず、無関係に名付けて良い。独自の呼び名。

```
def multiply(x, y):
    answer = x + y
    return answer
```

```
def multiply(a, b):
    answer = a + b
    return answer
```

関数定義を含むコードを実行する様子

sample_func.py

```
def multiply(x, y):
    answer = x * y
    return answer

width = 2
height = 5
area = multiply(width, height)
print(area) #-> 10
print(answer) #-> NameError
```

実行する様子

- multiplyという名前の関数を定義している。後から使うだろうから今はレシピを覚えておこう。（定義してるだけ。実行しない。）
- widthに2を保存。
- heightに5を保存。
- multiply関数を実行し、その戻り値をareaに保存。

ローカル変数とスコープ(scope)

sample_func.py

```
def multiply(x, y):
    answer = x * y
    return answer

width = 2
height = 5
area = multiply(width, height)
print(area) #-> 10
print(answer) #-> NameError
```

変数x, yは、関数multiply()の中でのみ使用できるローカル変数。関数の外からはアクセスできない。

・変数answerは、このスコープではまだ設定されていない。
・関数multiplyの中で設定している変数answerとは異なる。
・そのためNameError。

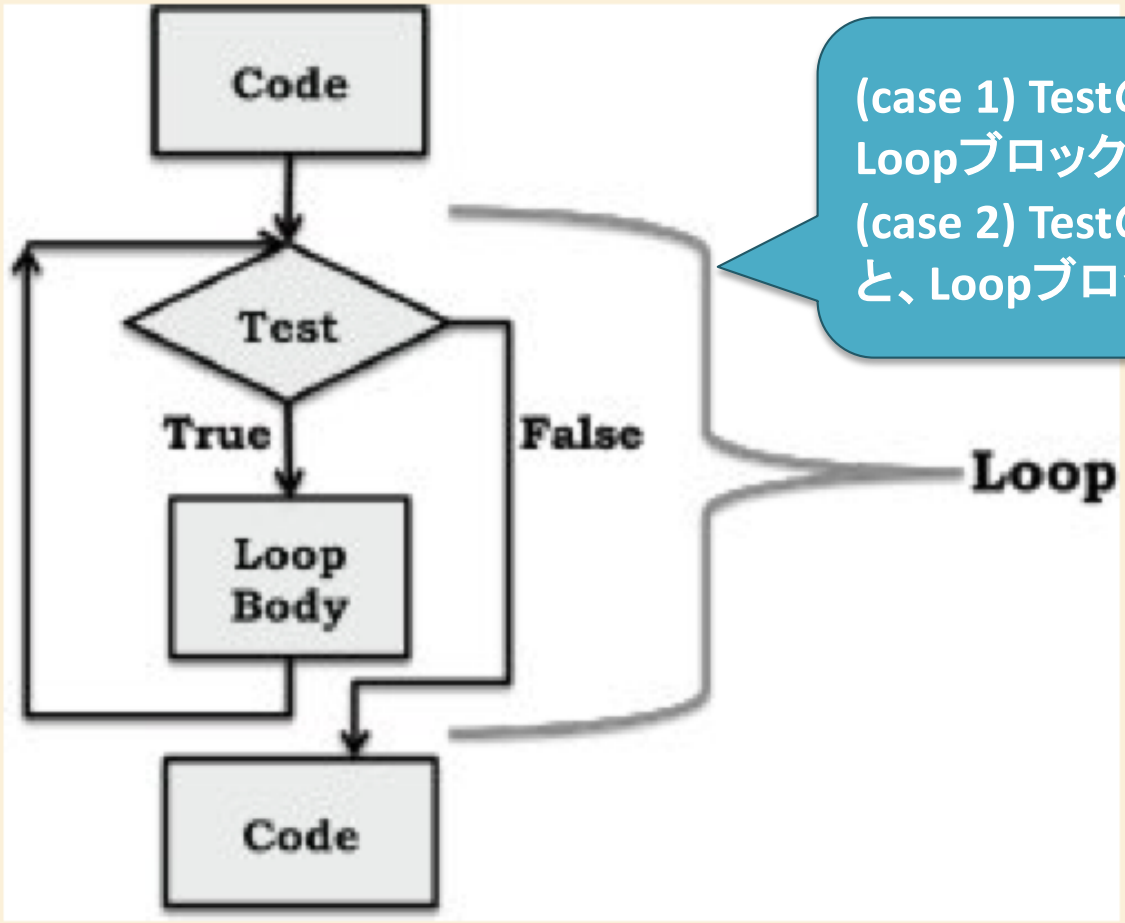
プログラミング1

(第4回) 関数の利用2、ループ処理（while文）

1. Chapter 4.1.1の補足2
    1. 関数とローカル変数
2. Chapter 3.1の補足
    1. Iteration, looping (反復処理)
    2. ループ処理の例、実行例
    3. 3種類の処理流れ制御
    4. ループ文の補足
3. 演習
4. 宿題

関数内の変数と、関数外の変数はスコープが異なることに注意。

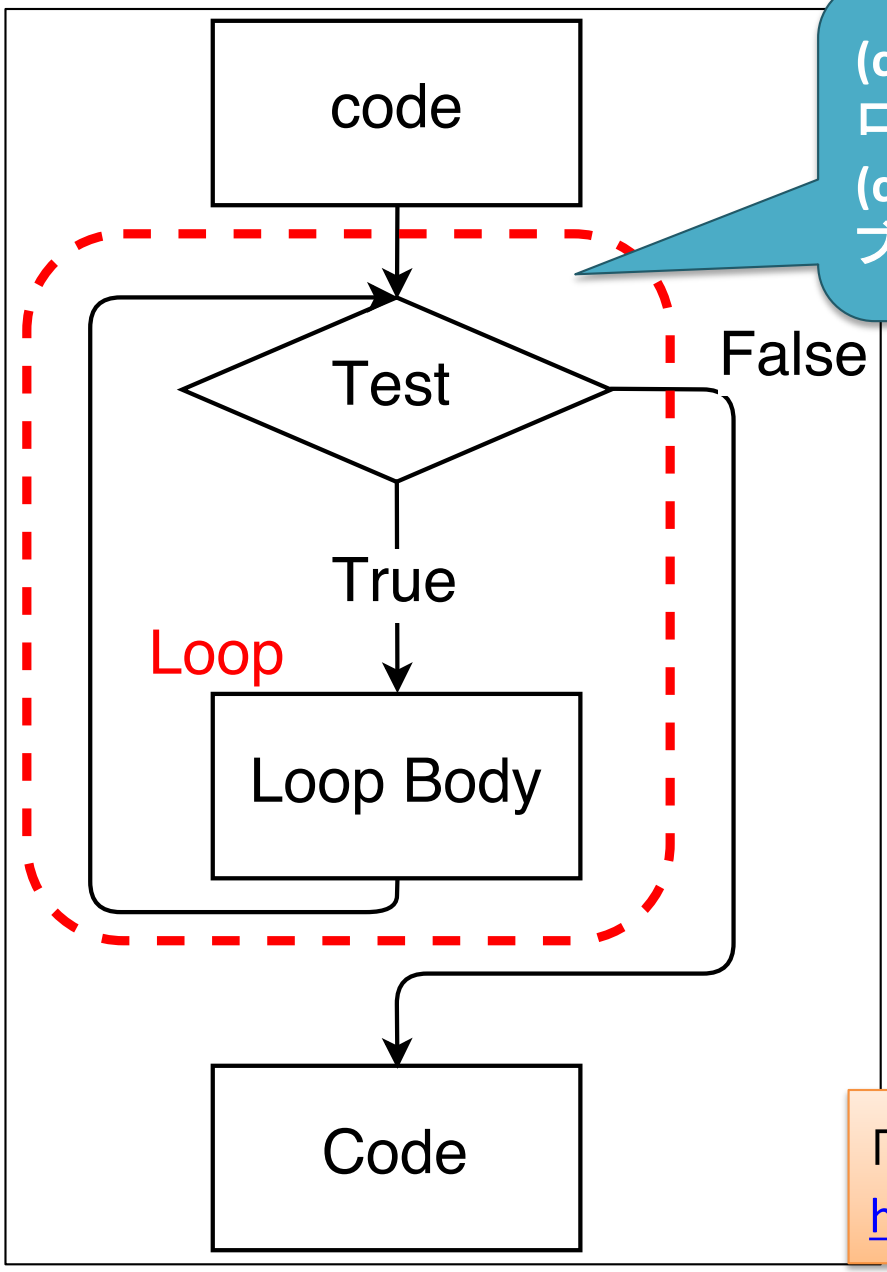
2.4 Iteration, looping (反復処理, 繰り返し処理)



(case 1) Testの結果がTrueの間、Loopブロックを繰り返す。
(case 2) Testの結果がFalseになると、Loopブロックを抜ける。

Figure 2.4 Flow chart for iteration

2.4 Iteration, looping (反復処理, 繰り返し処理)



(case 1) Testの結果がTrueの間、Loopブロックを繰り返す。
(case 2) Testの結果がFalseになると、Loopブロックを抜ける。

「draw.io」図描画webサービス
https://www.draw.io/

ループ処理の例 (2.4節の改良版)

```
# スライムのHPが0より大きい間タコ殴りにするゲーム
import random

def encount_enemy():
    hitpoint = random.randint(3, 7)
    return hitpoint

hp = encount_enemy()
print('敵に遭遇した。(敵HP={})!'.format(hp))

while (hp > 0):
    damage = random.randint(2,4)
    hp = hp - damage
    print('敵に{}のダメージ!(敵HP={})'.format(damage,hp))

print('スライムを倒した！')
```

乱数を用意してくれるツール(ライブラリ)。

HP3～7を取る敵を用意する関数を定義。この時点では、まだ実行されない点に注意。

定義した関数を使って、敵を用意。

「条件がTrueの間」＝「敵のHPが0より大きい間」、下記ブロックを繰り返す。

ループ処理の例 (2.4節の改良版)

```
# スライムのHPが0より大きい間タコ殴りにするゲーム

import random

def encount_enemy():
    hitpoint = random.randint(3, 7)
    return hitpoint


hp = encount_enemy()
print('敵に遭遇した。(敵HP={})!'.format(hp))


while (hp > 0):
    damage = random.randint(2,4)
    hp = hp - damage
    print('敵に{}のダメージ!(敵HP={})'.format(damage,hp))

print('スライムを倒した！')
```

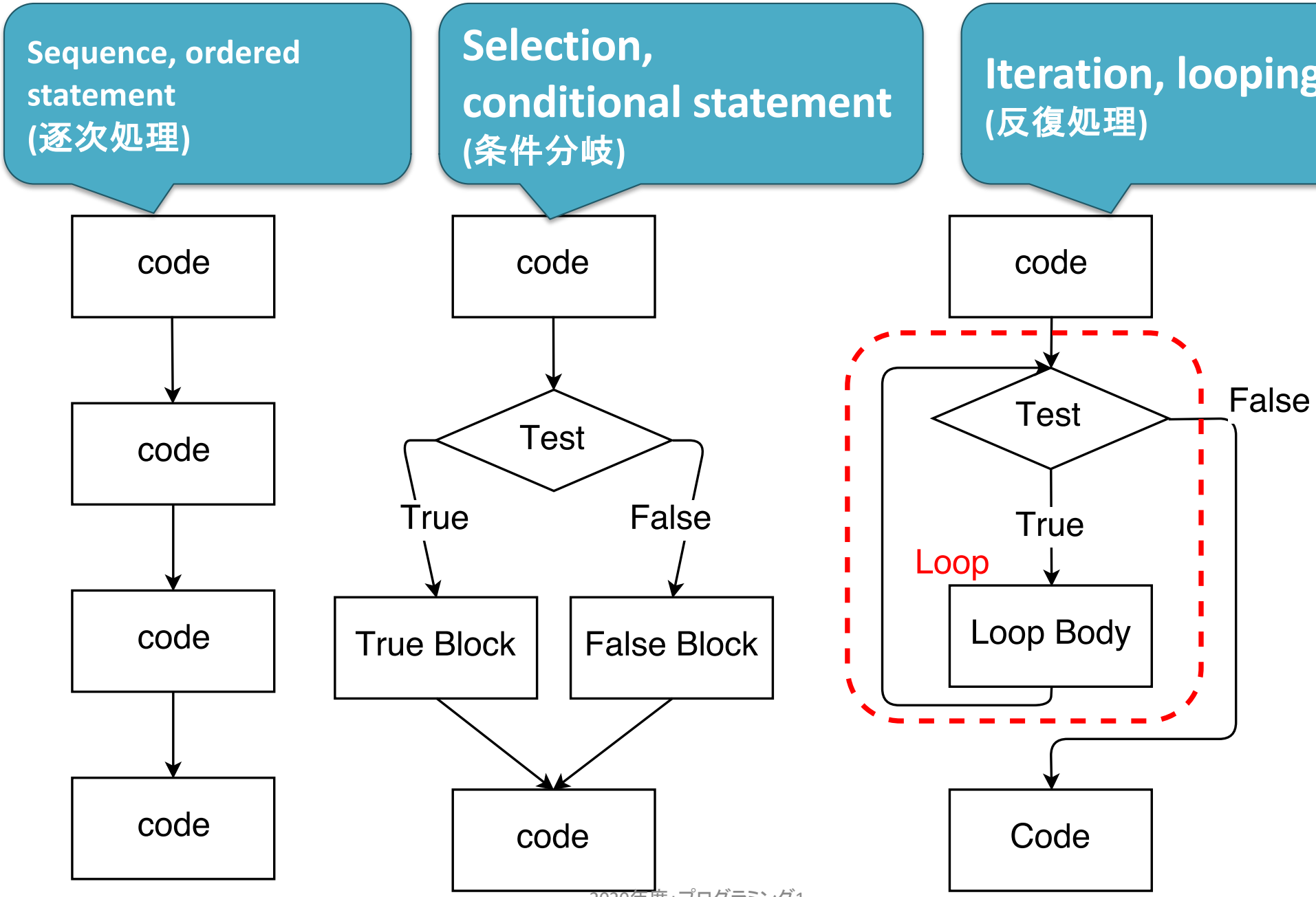
プログラミング1

(第4回) 関数の利用2、ループ処理（while文）

1. Chapter 4.1.1の補足2
    1. 関数とローカル変数
2. Chapter 3.1の補足
    1. Iteration, looping (反復処理)
    2. ループ処理の例、実行例
    3. 3種類の処理流れ制御
    4. ループ文の補足
3. 演習
4. 宿題

繰り返させたい処理をブロックで指定し、繰り返し条件を設定。

3種類の流れ制御 (control flow)



プログラミング1

(第4回) 関数の利用2、ループ処理（while文）

1. Chapter 4.1.1の補足2
    1. 関数とローカル変数
2. Chapter 3.1の補足
    1. Iteration, looping (反復処理)
    2. ループ処理の例、実行例
    3. 3種類の処理流れ制御
    4. ループ文の補足
3. 演習
4. 宿題

処理の流れは**逐次・条件分岐・反復処理**の3タイプのみ。

基本道具
- 型（int,float,str,bool）
- 演算(数値・文字列・比較・論理）
- フロー制御（if, while, for）
- 関数定義（def）

ループ文の補足

continue, break

Loop補足1（continue）

```
# レポート出した人（0点以外）の平均点を出したい。
scores = [80, 100, 0, 60]
sum = count = 0    # countはレポートを出した人数
for score in scores:
    if score == 0:
        continue
    sum += score
    count += 1

average = sum/count
print(sum, count, average)    # -> 240 3 80.0
```

デフォルトでは、スコアをsumに加算し、人数カウントを1増やす。ただし、score==0の場合には未提出扱いとして、加算処理もカウント処理もしたくない。この場合はループブロックの残った命令を処理せず、次のシーケンスに移り、次の要素に対して処理を継続（=continue）させたい。

Loop補足2（break）

```
# レポート出した人（0点以外）の平均点を出したい（バグ有り）
scores = [80, 100, 0, 60]
sum = count = 0    # countはレポートを出した人数
for score in scores:
    if score == 0:
        break
    sum += score
    count += 1

average = sum/count
print(sum, count, average)    # -> 180 2 90.0
```

先程と同様のコードで、continueをbreakに変更。
この場合、ループブロック自体から抜け出し（break）、その後のループ処理を終了する。

プログラミング1

(第4回) 関数の利用2、ループ処理（while文）

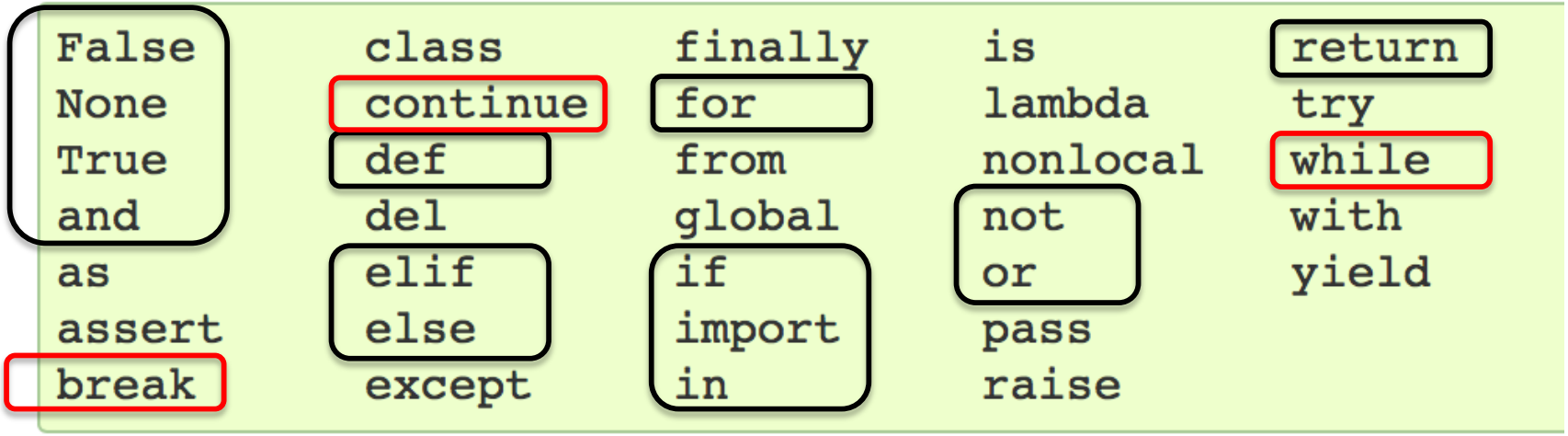
1. Chapter 4.1.1の補足2
    1. 関数とローカル変数
2. Chapter 3.1の補足
    1. Iteration, looping (反復処理)
    2. ループ処理の例、実行例
    3. 3種類の処理流れ制御
    4. ループ文の補足
3. 演習
4. 宿題

ループ処理を操作したいなら continue, break を使おう。

Reserved words, 予約語

https://goo.gl/rEzdAN

- 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・課題への取り組み方

取り組み方の例

- 問題を分割する。
  - 分割して分かるところから手を付ける（土台を作る）
  - 分からないところは、更に分割できないか考える
    - それでも分からないなら、分割の仕方や、分割した問題の解き方を訪ねてみよう
    - 「分割の仕方」＝問題解決手段の一つ
- 個々のサブ問題を個別に解く。
  - これ以上分割できない≒最小の部品なら教科書・授業で習ってるはず
    - ->該当部分を復習
      - 該当部分が分からないなら、該当部分の探し方を尋ねてみよう。
      - 教科書・授業の復習が足りてないかも。
- それらの組み合わせ方を考える。

演習

演習1～4: 初めてのレポート

プログラミング1

(第4回) 関数の利用2、ループ処理(while文)

1. Chapter 4.1.1の補足2
    1. 関数とローカル変数
2. Chapter 3.1の補足
    1. Iteration, looping (反復処理)
    2. ループ処理の例、実行例
    3. 3種類の処理流れ制御
    4. ループ文の補足
3. 演習
4. 宿題

関数内の変数と、関数外の変数はスコープが異なることに注意。

繰り返させたい処理をブロックで指定し、繰り返し条件を設定。

処理の流れは逐次・条件分岐・反復処理の3タイプのみ。

ループ処理を操作したいなら continue, break を使おう。

基本道具
- 型(int,float,str,bool)
- 演算(数値・文字列・比較・論理)
- フロー制御(if, while, for)
- 関数定義(def)

宿題

- 復習: 適宜(これまでの内容)
- 予習: 教科書読み
  - 3章
    - 3.2 For Loops
    - (スキップ) 3.3 Approximate Solutions and Bisection Search
    - 3.4 A Few Words About Using Floats
- 復習・予習(オススメ): progate

参考文献

- 教科書: Introduction to Computation and Programming Using Python: With Application to Understanding Data
- Python 3.7 documentation, https://docs.python.org/3.7/index.html
- Google Python Style Guide, https://google.github.io/styleguide/pyguide.html