人工知能
IE229 – ARTIFICIAL INTELLIGENCE

第5回講義: Python Programming (1)
Lecture 5: Python Programming (1)

二宮 崇 (Takashi Ninomiya)
愛媛大学 (Ehime University)
ninomiya@cs.ehime-u.ac.jp

Python演習

- Python https://www.python.jp/
  - 最もよく使われているプログラミング言語の一つ
    - 多くのプログラミング言語ランキングで1位～2位。
  - ライブラリが充実しており、データ処理、データ解析、機械学習のためによく使われる。特に深層学習のプログラムはPythonで書くことが多い。
  - ドキュメント https://docs.python.jp/3/

※Pythonには2系と3系があり、構文や同じ関数でも処理が違う場合があるので注意。本講義ではPython 3系を想定。

Python環境設定

- **深層学習(ディープラーニング)を行うには？**
  - クラウド環境を使う方法
    - Google Colaboratory (略称 Colab)
      - https://colab.research.google.com/
      - Googleが提供するクラウド環境の深層学習プラットフォーム
      - Googleアカウントがあれば無料で制限付きながら使える
      - Googleアカウントをもっていればおすすめ
    - Microsoft Azure Machine Learning (通称 Azure)
      - Microsoftアカウント+サブスクライブで使える。
      - 無料で1年間だけ制限付きながら使える。

Python環境設定

- **深層学習(ディープラーニング)を行うには？**
  - 自分が持っているPCを使う方法
    - Pythonに自力でいろんなライブラリを追加する
      - バージョンの違いでうまくインストールできなかったり動かなかったりでとにかく大変なのであまりおすすめできない。
    - Anacondaを使う (オススメ！)
      - Python + 科学技術計算(データサイエンス)ライブラリ
      - データサイエンス向けライブラリが全部入りで最初から入っている
      - 最初から入っているライブラリ
        - NumPy ベクトル行列演算ライブラリ
        - SciPy 数値演算、最適化、信号処理、関数
        - Pandas 統計処理、データ分析、時系列解析
        - scikit-learn 機械学習

Python環境設定

- **この授業では次のプログラミング環境を使って演習を行います**
  - Google Colaboratory

https://colab.research.google.com

ノートブックを新規作成で新しくノートブックを作ります。

  - PyTorch (深層学習ライブラリ)
    - Google Colaboratoryにはすでに入っています。

Import torch

で実行可能になります。

Python演習

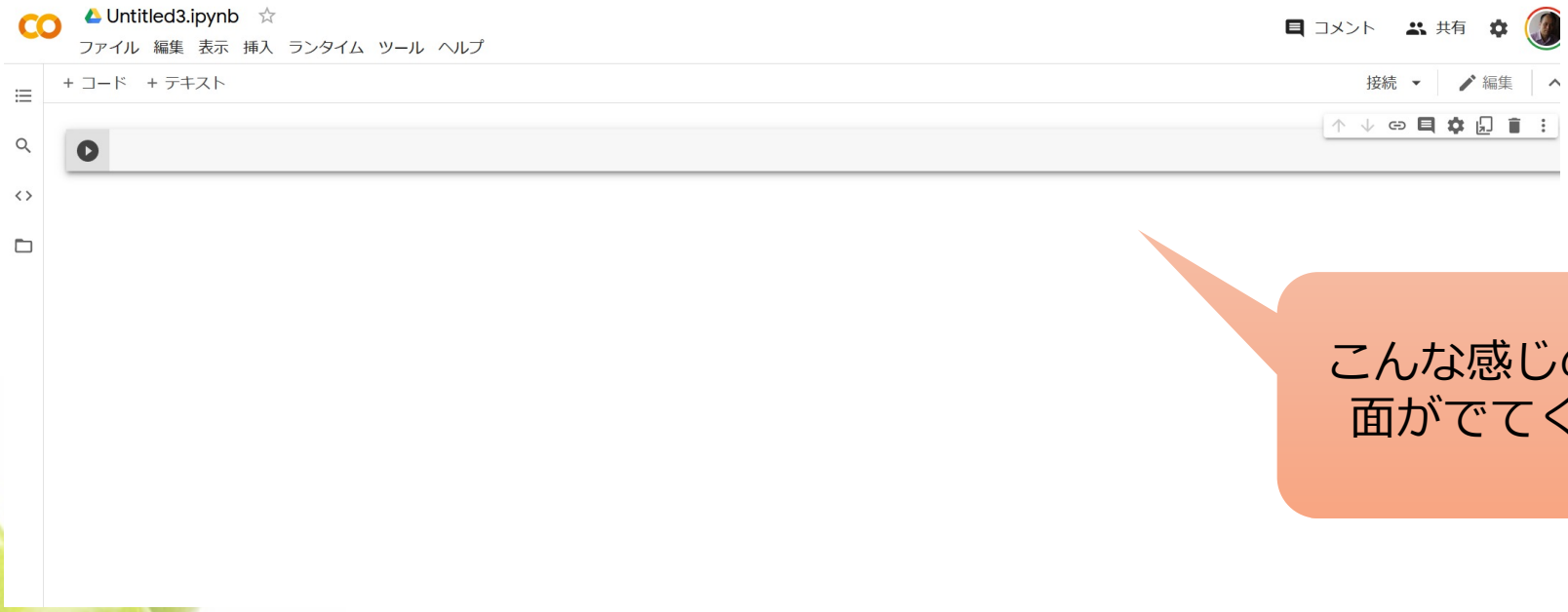
PYTHONプログラミング

ノートブックの新規作成

- Google Colaboratoryでノートブックを新規作成します。

Jupyter Notebook

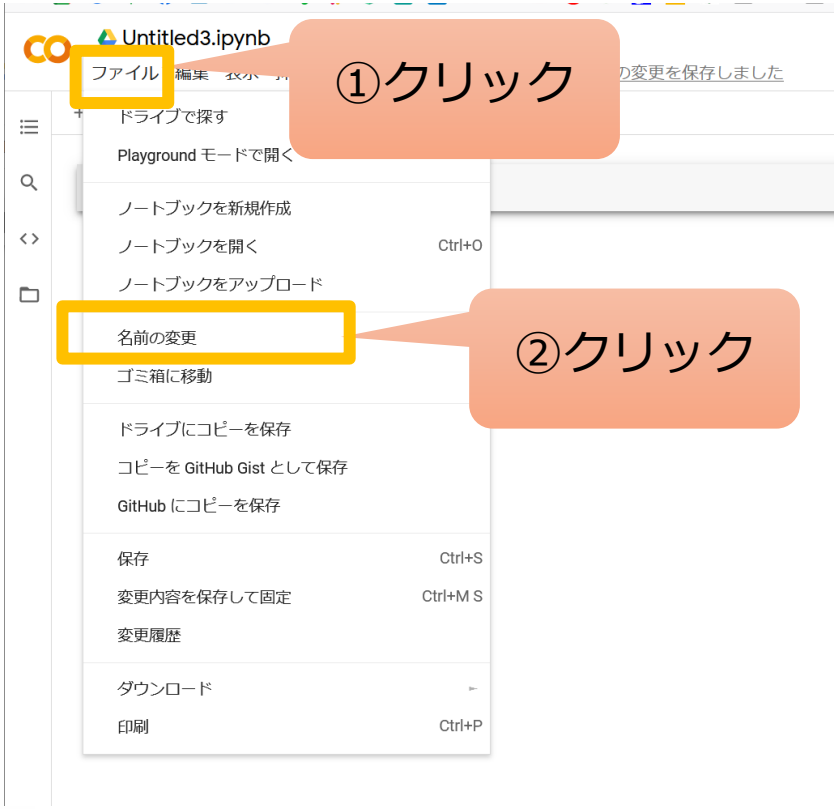
- **Jupyter Notebookと呼ばれるプログラミング環境が用意されます。**
  - ファイルにはコードとテキストをいれることができます。
  - Jupyter Notebookのファイル拡張子は.ipynb



こんな感じの画面がでてくる

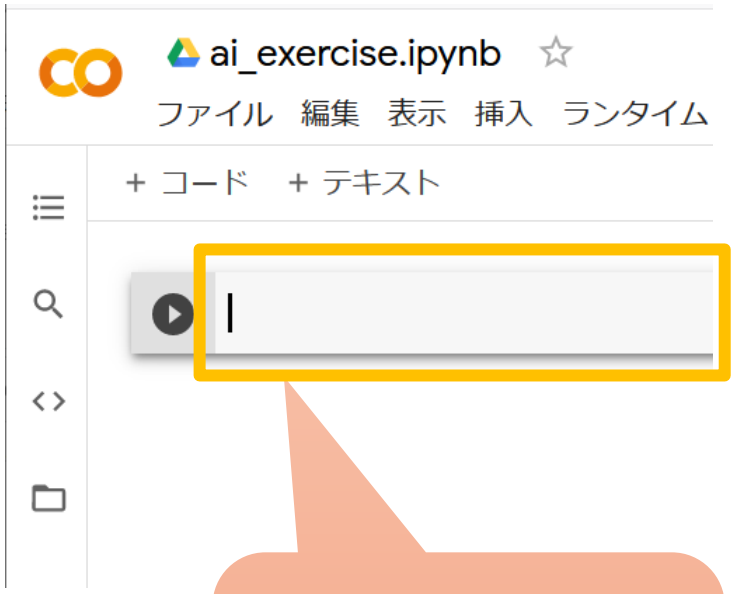
ファイル名の変更

- まず、ファイル名を変更しましょう。(好きな名前で良いです。ai5_exercise.ipynb、ai5_practice.ipynbなど。演習用のファイル名とわかるような名前にしましょう)

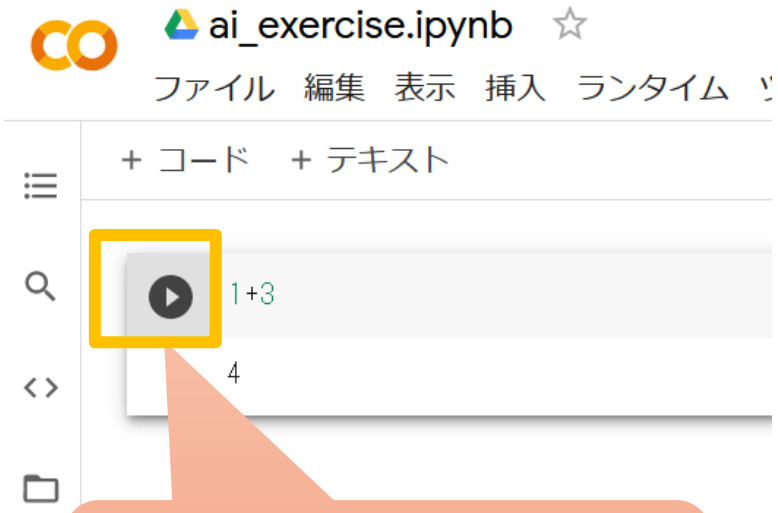


コードを書こう

- **コード(プログラム)を書こう**



①ここにプログラムを書いていくことができます。

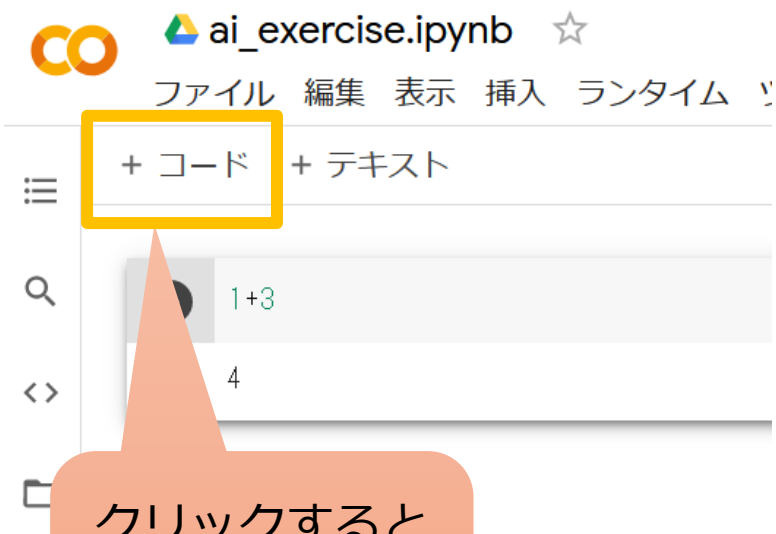


②クリックすることでこのセルのプログラムを実行

(参考)
Ctrl+Enter: セルの実行

コードを追加しよう

- **+コードをクリックするとコードを追加できる**



クリックするとコードを書くセルを追加できる

(参考)
Alt+Enter: セルの実行+コードの追加

Python 算術演算

- **Pythonでの算術演算**
  - 足し算 +
  - 引き算 –
  - 掛け算 *
  - 割り算 /
  - 整数の割り算 //
  - 余りの計算 %
  - べき乗 **
- **右のプログラムコードを実行してみよう**

```
[1] 1+3
4

[2] 3-1
2

[3] 5*2
10

[4] 2**4
16

[5] 7/5
1.4

[6] 7//5
1

[7] 7%5
2
```

Python 変数

- アルファベットで定義
- 整数型(int)や実数型(float)といった型は宣言せず自動的に決定される
- 変数への代入は「=」
- 変数の型や計算結果の型はtype関数で調べることができる
- 右のプログラムを実行してみよう

```
[8]  x=100

[9]  x

     100

[10] type(x)

     int

[11] y=3.14

[12] type(y)

     float

[13] x*y

     314.0

[14] type(x*y)

     float
```

変数の中身をみることができる

intは整数型という意味

floatは実数型という意味

計算結果にも型がついている

Python 変数

- **次のプログラムを実行してみよう**

```
[15] a=1

[16] a
     1

[17] type(a)
     int

[18] b=1.0

[19] type(b)
     float
```

小数点をつけない数字は整数型と判断される

実数型にするには小数点をつけて入力する

```
[20] c=1.

[21] c
     1.0

[22] type(c)
     float
```

実数型をこのように入力しても良い

ブール型

- **ブール型(bool): TrueかFalseの値**
- **ブール演算**
  - and演算: $x\ and\ y = \begin{cases} True & xとyの両方ともTrueのとき \\ False & それ以外 \end{cases}$
  - or演算: $x\ or\ y =$
    $\begin{cases} True & xとyのどちらかもしくは両方がTrueのとき \\ False & それ以外 \end{cases}$
  - not演算: $not\ x = \begin{cases} True & xがFalse \\ False & xがTrue \end{cases}$ (xのTrue/Falseの反転)

ブール型

- コンソールで次のPythonプログラムを書いてみよう

```
[23] hungry=True

[24] sleepy=False

[25] type(hungry)

     bool

[26] not hungry

     False

[27] hungry and sleepy

     False

[28] hungry or sleepy

     True
```

文字列型

- **文字列(string)**
  - '(シングルクォート)で囲む
  - "(ダブルクォート)で囲む
  - """(ダブルクォート3つ)で囲む
  - どの方法でも同じ文字列が得られる
- **文字列の連結は「+」で実行できる**
- **右のプログラムを実行してみよう**

```
[29] a = "こんにちは"

[30] b = 'こんにちは'

[31] c = """こんにちは"""

[32] a

     'こんにちは'

[33] b

     'こんにちは'

[34] c

     'こんにちは'

[35] a+b+c

     'こんにちはこんにちはこんにちは'
```

文字列の連結

リスト型

- **リスト型(list)**
  - 複数のデータをまとめた並び
  - カンマ区切りの値の並びを角括弧で囲む

```
[36] a = [1, 2, 3, 4, 5]

[37] a
     [1, 2, 3, 4, 5]

[38] len(a)
     5

[39] a[0]
     1

[40] a[1]
     2
```

角括弧の中にリストの各要素を並べる。各要素はカンマ(,)で区切る

リストの長さは**len関数**で計算できる

a[n]と書くことでn番目の要素を得ることができる。ただし、リストの最初の要素は0番目と数える

```
[41] a[4]
     5

[42] a[4]=99

[43] a
     [1, 2, 3, 4, 99]
```

a[n]=xと書くことでn番目の要素にxを代入することができる

リスト型

- **リストの連結**
  - +でリストを連結することができる（文字列の連結と似ている）

リストの連結

```
[44] a=[1,2,3]

[45] b=[4,5,6]

[46] a+b

     [1, 2, 3, 4, 5, 6]
```

- **リストの生成**
  - [x]*n でxを要素とする長さnのリストを生成できる

要素1の長さ10のリスト

```
[47] a=[1]*10

[48] a

     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

辞書型

- **辞書型(dictionary): 一般には連想配列と呼ばれるデータ構造**
  - キーと値のペアをたくさん格納したリスト
  - キーに対応する値をすぐに取り出すことができる
  - 「キー:値」を並べて中括弧で囲むことで生成する

```
[49] height={"sato":180, "yamada":170}

[50] height["sato"]
     180

[51] height["kato"]=175

[52] height
     {'kato': 175, 'sato': 180, 'yamada': 170}
```

”sato”は180
”yamada”は170

d[x]は辞書dにおけるxの値

d[x]=yで、辞書dにおけるxの値をyにする

”kato”:175が追加されている

if文

- **条件分岐**
  - ブロック: 一連の処理の範囲のことをブロックという。
  - CやProcessingなど普通の言語では中括弧{}を用いてブロックを指定する。
    - Processingの例

```
int a = 150;
if( a > 100) {
  background(255,0,0);
  ellipse(50, 50, 50, 50);
}
```

If文の条件式が真であったときのブロック

  - Pythonのブロックはインデントで指定
    - Pythonの例

```
[53] a = 150

[54] if a > 100:
       a = 0
       print("1")
     else:
       print("0")

     1
```

If文の条件式が真であったときのブロック

空白4個かタブが一般的

If文の条件式が偽であったときのブロック

if文

- **次のコードを実行してみよう**

```
[55] a=150
     if a > 100:
       print("aは100より大きい")
     elif a > 0:
       print("aは0より大きくて100以下です")
     else:
       print("aは0以下です")
```

aの値をいろいろと変えてどのように結果が変わるのかみてみよう

```
aは100より大きい
```

if文の構文

条件式の後にコロン(:)が必要

```
if 条件式1:
        条件式1が真のときの処理
elif 条件式2:
        条件式1が偽で条件式2が真だったときの処理
…
else:
        上記の条件式がすべて偽だったときの処理
```

上から下に向かって実行

if文の条件式

- **条件式には次のようなものが使えます**
  - 等号(等しい) ==
  - 不等号(より大きい) >
  - 不等号(より小さい) <
  - 不等号(以上) >=
  - 不等号(以下) <=
- **また、ブール演算を使って、複雑な条件を書くこともできます**
  - 例

```
[56] x=150
     if 100 < x and x <= 200:
       print("xは100より大きく200以下です")

     xは100より大きく200以下です
```

for文

- Pythonのfor文はリストに対する繰り返し処理

```
[57] words = ["cat", "dog", "lion"]
     a=1
     for w in words:
       print(w+": "+str(a))
       a = a + 1
```

```
cat: 1
dog: 2
lion: 3
```

for文の構文

```
for 変数 in リスト:
        繰り返し処理のブロック
        （変数を参照しながら処理することができる）
```

for文

- Pythonのfor文はリストに対する繰り返し処理

```
[57]  words = ["cat", "dog", "lion"]
      a=1
      for w in words:
          print(w+": "+str(a))
          a = a + 1
```

①　②　③

繰り返し処理ブロック

文字列と数値の足し算はできない。
str関数を用いると数値を文字列に変換できる。

①に対する処理　②に対する処理　③に対する処理

リストの先頭から順に各要素が変数に代入されて繰り返し処理ブロックの処理が行われる

①w = "cat"として繰り返し処理ブロックを実行
②w = "dog"として繰り返し処理ブロックを実行
③w = "lion"として繰り返し処理ブロックを実行

for文とrange関数

- **一定回数繰り返したいときは？→range関数を使う**

- **range関数**
  - range(n)で[0, 1, 2, …, n-1]の(仮想的な)リストを作る
  - range(i, j)で[i, i+1, i+2, …, j-1]の(仮想的な)リストを作る

```
[58] list(range(20))

     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
[59] list(range(10, 20))

     [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

for文とrange関数

- for文とrange関数を用いた繰り返し処理

```
[60] sum=0
     for i in range(100):
       x = i + 1
       sum = sum + x
     print(sum)
```

[0, 1, 2, …, 99]のリストがあると思えば良い

x = i + 1とすることで、xは1, 2, …, 100の値になる

5050

i = 0 としてx = 1, sum = sum + x
i = 1 としてx = 2, sum = sum + x
i = 2 としてx = 3, sum = sum + x
…
i = 99としてx = 100, sum = sum + x

これは$\sum_{x=1}^{100} x$ =1 + 2 + 3 + … + 100=5050の計算をしている

関数

- lenなど組み込みの関数を用いてきたが自分で新しく関数を定義することができる
- defで関数を定義

```
[61] def add(x, y):
        ans = x + y
        return ans

     print(add(10,30))
```

```
40
```

関数定義の構文

```
def 関数名(引数の列):
        関数内での処理
        return 返り値
        (返り値は関数が返す値)
```

Python演習

- 次のプログラムを入力して実行してみよう

```
[62] def sign(x):
       if x < 0:
         print("negative")
         return -1
       elif x == 0:
         print("zero")
         return 0
       else:
         print("positive")
         return 1

     x = sign(10)
     y = sign(0)
     z = sign(-5.3)
     print(x, y, z)
```

```
positive
zero
negative
1 0 -1
```

このような結果がでていたらうまく動いているということ

Python演習

- **数値のリストを受け取ったとき、受け取ったリストの各要素を10倍にして返す関数を書きましょう。**
  - まず、「ファイル」→「ノートブックを新規作成」を選んで新しいファイルを作成しましょう。
  - 続いて、新しく作ったファイル名を変更しましょう。(例えば、ai5_tentimes.ipynbなど)

```
def tentimes(numlist):
  #ここにプログラムを書く


print(tentimes([1,2,3,4,5]))
```

```
[10, 20, 30, 40, 50]
```

うまくプログラムが書けたらこのような結果が返ってくる

Python演習 解答例

- **解答例**

```
def tentimes(numlist):
  r = []
  for x in numlist:
    r = r + [10*x]
  return r

print(tentimes([1,2,3,4,5]))
```

最初にrに空リストをいれる

rにnumlistの各要素を10倍にした値を追加していく

最後にrを返す

リストの連結はリスト同士でないとできないので、要素1個のリスト[10*x]を作ってから連結している

Python演習 解答例

- **別解**

```
def tentimes(numlist):
  r = [0]*len(numlist)
  for i in range(len(numlist)):
    r[i] = 10 * numlist[i]
  return r

print(tentimes([1,2,3,4,5]))
```

最初にnumlistと同じ長さのダミーのリストを作っておく

i=0, .., (numlistの大きさ-1)まで繰り返す

r[i]をnumlist[i]の10倍の値にする

最後にrを返す

タプル型

- **タプル型（tuple）**
  - 複数のデータをまとめた並び
  - カンマ区切りの値の並びを丸括弧で囲む(正確には、カンマ区切りで並べたものがタプルとなり、丸括弧は省略可。)

  x = ('cat', 'dog', 'lion')　　→ x = 'cat', 'dog', 'lion'と書いても同じ
- **リスト型との違い**
  - 追加・変更・削除ができない
    - リストより不便だけど効率が良い。辞書型のキーに適している
  - 丸括弧をつける必要がない
    - カンマ区切りで並べたものはタプルとなる。
    - 簡便に書くことができる
    - 関数において値をまとめて返したり、変数をまとめて代入するときによく使う

    return x, y　　→ xとyをタプルとして返す。return (3, 4)と書くのと同じ。

    a, b = 3, 4　　→ a = 3, b = 4となる。(a, b) = (3, 4)と書くのと同じ。

    x, y = y, x　　→ xとyの入れ替え(スワップ)

内包表記

- **内包表記 (comprehension)**

[式e for 変数x in リストl]

  - for文を用いたリストの生成を簡便に行うための構文
  - リストlの各要素を変数xに代入し、式eを実行する
  - 式eの実行結果をリストとして順に並べて出力する

  - 例

    [10*x for x in [1,2,3,4]]

    →[10, 20, 30, 40]

ラムダ式

- **ラムダ式**

lambda 変数x: 式e

- 名前無し関数を生成
- 変数xを受け取って、式eの計算結果を返す関数
- 例 f = lambda x: x+10

- 次の 2 つは同じ定義

```
f=lambda x: x+10
```

```
def f(x):
    return x + 10
```

3項演算 (if関数)

- **Pythonにおいて通常用いられるifは制御構造文(返値がない)であったが、if関数(3項演算子)も用意されている**

式e if 条件式c else 式f

  - 条件式cを満たす時、式eの計算結果を返す。そうでなければ、式fの計算結果を返す

  - 例 x = 'odd' if y % 2 == 1 else 'even'

まとめ

- **Python**
  - Colaboratoryでのプログラムの書き方
  - 算術演算
  - 変数
  - ブール型、文字列型、リスト型、辞書型
  - if文、for文とrange関数
  - 関数
  - タプル型、内包表記、ラムダ式、3項演算