一般教育演習

プログラミングで問題を解く：集計から人工知能まで

瀧川 一学
工学部 情報理工学コース

今日のお題：ガイダンス！

- **テクニカルなこと**

  どういうことが学べるか，どう実施するか，成績は？

- **講師自己紹介**

  わたしとプログラミング

- **プログラミング漫談**

  プログラミングを取り巻く最近の事情？

出席

- 出欠確認システムの情報を使います．各自，学生証をICカードリーダーに入室時と退出時の2回かざしてください.
- 学生証を忘れた場合，授業開始前に学生番号と名前を報告してください．それ以外ではいかなる理由でも出欠確認システムの情報は訂正できません.

授業の目標 (シラバスより)

プログラミングとは何だろうか？ある人はそれは「アート」な知的活動だと言い、ある人はそれはコンピュータに魔法をかけるための「呪文」だと言う。今や私たちの社会や生活は、電子機器を動かしたりデータをやり取りする無数のコンピュータプログラムが支えている。プログラミングとはそれらのコンピュータプログラムを作る活動である。この入門授業では、実際にプログラムを作って動かすという実習とフォローアップディスカッションを通してプログラミングとは何かを理解することを目標とする。この「アート」な「呪文」に関心があるなら、文系理系やプログラミングの経験は全く問わない。

到達目標 (シラバスより)

1. 「プログラミングとは何か？」について自分なりの答えが持てるようになる。

2. プログラミングに最低限必要な「コンピュータの仕組み」や「アルゴリズム」「データ構造」と言った初歩的知識を身につける。

3. 実際にプログラミング言語を用いて簡単なプログラムを作成するための基本的なスキルを身につけ実践することができる。

4. 今後、プログラミングを自ら習得していくために何をしていけば良いかの理解や知識を得る。

5. コンピュータプログラムが我々の生活社会の基盤にあることを意識し、品質の高い美しいプログラムを作り上げることの価値を議論できるようになる。

授業計画 (シラバスより)

1. ガイダンス：授業の進め方を説明
2. プログラミングとは？(1)：自己紹介～コンピュータの仕組みとプログラム、その社会における役割
3. プログラミングとは？(2)：実習環境設定～初めてのプログラミング
4. 基本文法を学ぼう(1)
5. 基本文法を学ぼう(2)＋フォローアップ
6. 標準ライブラリを使ってみよう(1)
7. 標準ライブラリを使ってみよう(2)＋フォローアップ
8. プログラムの構造と設計(1)
9. プログラムの構造と設計(2)＋フォローアップ
10. データを読み解こう(1)
11. データを読み解こう(2)＋フォローアップ
12. 品質の高い美しいプログラムを作るには(「アート」な「呪文」への道)＋ディスカッション
13. 画像を認識するAIプログラムを作ってみよう(1)
14. 画像を認識するAIプログラムを作ってみよう(2)
15. 授業のまとめと振り返り：この後どうする？

対象者

プログラミングに興味があるが、

(1)ほとんど、あるいは、全くしたことがない人

(2)してみたが理解がふわっとしていてよくわからなかった人

(3)したことがあるが自信がないので考え直したい人

(4)既にものすごくできるがなぜか急に入門に参加してみたくなった人

　(プログラミングを学ぼうとする同志をサポートしたい人)

(5)その他すべての人

大事なこと

注意：これは「プログラミング」を教えてもらう受動的座学の授業ではなく，「プログラミング」をどうやって学んでいけば良いかの根本骨格，つまり自分で学んでいくための素養を得るための授業

なぜなら，

「プログラミング」とは「プログラミング言語」を学ぶことではないから

「プログラミング」とは結局自分で実際にやってみないと分からないから

例え：スイカの味を食べたことのない人にいくら説明しても結局食べてみないことには本当のことはわからない！

授業形態と成績について

- 出席2/3(10回)以上を成績評価対象
  (演習なので基本的には全ての授業への参加が望ましい)

- **成績**：授業や演習への積極性(30%), レポートや成果物の内容(50%)、ディスカッションへの貢献(20%)を総合的に評価

- **評価**：入門授業であるため、プログラムのクオリティそのものよりも、考察やコメントによる思考過程のプレゼンテーションをより重視する。プログラミングとは何かを理解し、プログラムが作れるようになっていく「学習過程の磨き方そのもの」を最も評価する。

成績について

- 同一科目が多数開講されることから，公平性と信頼性を確保するため評価は新GPA制度の「学修成果の質」に基づいて行い，可能なかぎり平均GPAが3.0±0.3の範囲内に収まるように，となっています.
- 成績評価はA＋からFまでの11段階で，A+は履修者の5%以内
- 出席の有無自体を点数化して評価に用いることはできないことになっています．(最低基準以上の出席が成績評価を受けるための必要条件)

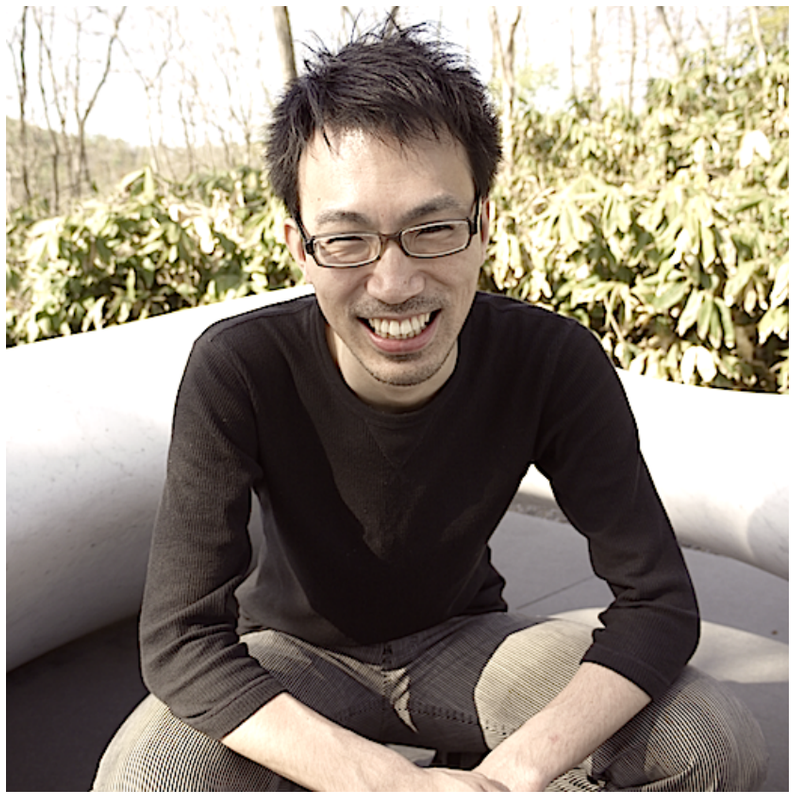
予定

- 授業や演習への積極性(30%)
- レポートや成果物の内容(50%)
- ディスカッションへの貢献(20%)

授業では実際にプログラミングをしてみるとともに「分からない点，不思議な点，もっと掘り下げたい点」を適宜フィードバックしてもらい，その回答を次回授業に反映(フォローアップ・ディスカッション).
後半は各々の興味で自由制作や調査なども.

瀧川 一学（たきがわ・いちがく）

プログラミングやAIど真ん中！？



- 北海道大学 情報科学研究科 情報理工学専攻
  大規模知識処理研究室（**工学部情報理工コース**）
  准教授
- 北海道大学 電子科学研究所
  附属社会創造数学研究センター
- 科学技術振興機構(JST)
  戦略的創造研究推進事業(さきがけ)研究者
  (マテリアルズインフォマティクス領域)
- 人工知能学会 人工知能基本問題研究会 主査

瀧川 一学 (たきがわ・いちがく)

北大で学ぶ (10年) 内訳: 大学4, 大学院2+3, 研究員1

😊多変量解析・統計的信号処理 (音響信号の分離技術)

京大で働く (7年) 内訳: 助教7

化学研究所（バイオインフォマティクスセンター）

薬学研究科（医薬創成情報科学専攻）

😊生命科学データからの知識発見・計算生物学

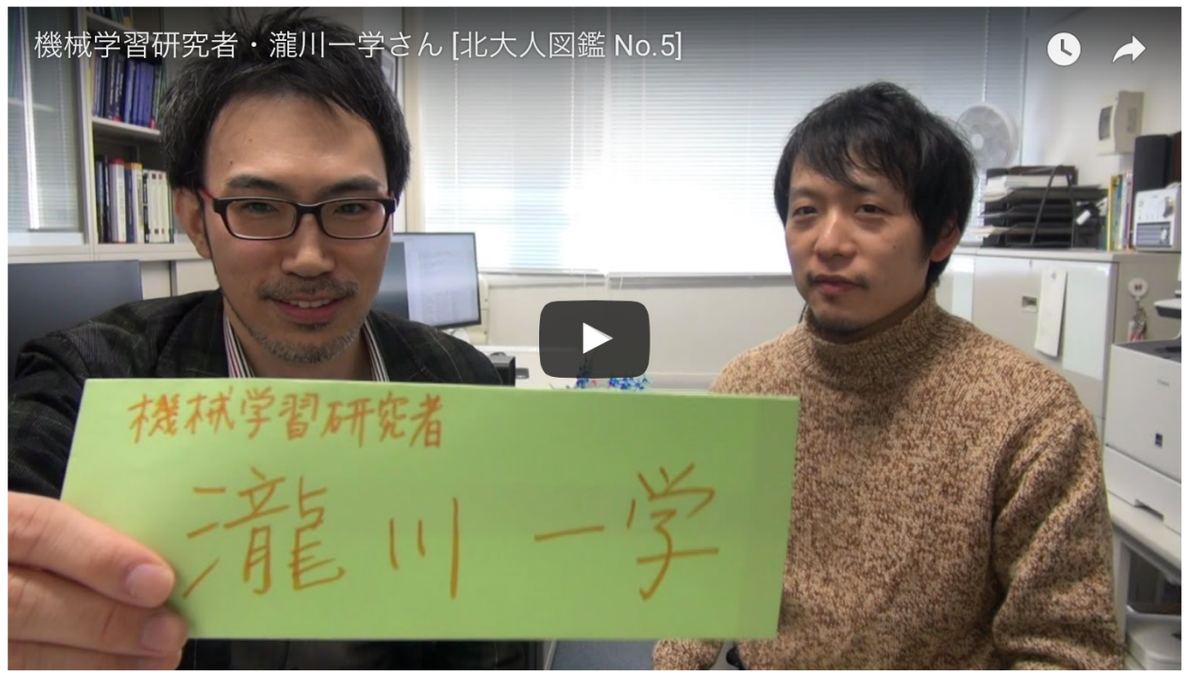
北大で働く (5.5年) 内訳: 特任助教2.5, 准教授3

😆離散構造を伴う機械学習、科学への機械学習の適用

瀧川 一学（たきがわ・いちがく）

http://art.ist.hokudai.ac.jp/~takigawa/

いいね！Hokudai　YouTube JP



[北大人図鑑 No.5]

機械学習研究者・瀧川一学さん

情報科学研究科 情報理工学専攻
准教授

３つのキーフレーズ：

1．ネットワーク構造をともなう機械学習
　　～大規模データが異分野の境界領域をつなぐ～
2．ドイツ、音楽、娘の成長から感じるAIの哲学とは？
3．熟練の経験と勘を、だれもが使えるツールに

プログラミングって何？

プログラミングとは何でないか

- 「コーディング」(コンピュータにむかって何やらかちゃかちゃ打ち込むこと！)とは違う
- 「プログラミング言語」の学習とは違う
- コンピュータを使ってやることというわけでもない(コンピュータなくてもプログラミングはできる)

答えは千差万別(みなさんの中に!)

瀧川の個人的感覚

「プログラミング」

＝ 論理的思考(設計)そのもの

＋表現や構造の美学

コンピュータが
実行するから
(工学技術)

人に読まれるため
に書くものだから
(創作活動)

到達目標 (シラバスより)

1. **「プログラミングとは何か？」について自分なりの答えが持てるようになる。**

2. プログラミングに最低限必要な「コンピュータの仕組み」や「アルゴリズム」「データ構造」と言った初歩的知識を身につける。

3. 実際にプログラミング言語を用いて簡単なプログラムを作成するための基本的なスキルを身につけ実践することができる。

4. 今後、プログラミングを自ら習得していくために何をしていけば良いかの理解や知識を得る。

5. コンピュータプログラムが我々の生活社会の基盤にあることを意識し、品質の高い美しいプログラムを作り上げることの価値を議論できるようになる。

2020問題?

2020年度（平成32年度）から小学校でプログラミング教育が必修化

教育の情報化の推進

小学校プログラミング教育の手引（第一版）

小学校プログラミング教育の手引（第一版）について

今般文部科学省では、新小学校学習指導要領におけるプログラミング教育の円滑な実施に向けて、「小学校プログラミング教育の手引（第一版）」を取りまとめました。

本手引は、学習指導要領や同解説で示している小学校段階のプログラミング教育についての基本的な考え方などをわかりやすく解説し、教師がプログラミング教育に対して抱いている不安を解消し、安心して取り組めるようにすることをねらいとしており、小学校プログラミング教育導入の経緯、小学校プログラミング教育で育む力、プログラミング教育のねらいを実現するためのカリキュラム・マネジメントの重要性と取組例などについて解説するとともに、教育課程内における指導例や、企業・団体や地域等との連携の例などを掲載しています。

また、本手引は、今後の各学校における実践の充実を踏まえ、また、「未来の学びコンソーシアム」の取組とも連携を図りながら、適時改訂を重ね、充実させていく予定です。

この手引を参照いただき、小学校段階のプログラミング教育の実施に向けての準備や実践等にお役立てください。

小学校プログラミング教育の手引（第一版）

- 小学校プログラミング教育の手引（第一版）　（PDF:3578KB）

- 教育の情報化の推進
- 教育の情報化に関する方針等
- 教育の情報化に関する取組
- 教育の情報化に関する基盤整備
- 社会教育の推進
- メディア教育
- リンク
- サイトマップ
- このウェブサイトについて

2020年度（平成32年度）から小学校でプログラミング教育が必修化

- 皆さんのお子さんはプログラミングを小学校で学ぶ (たぶん)
- 今生まれている子供の保護者たちも戦々恐々 (娘2歳7ヶ月)
- 「プログラミング」という教科が作られるわけではない
- プログラミングスクールや早期教育がもてはやされている？
- 一因に「人工知能(AI)技術」の社会浸透？

子どもを
億万長者に
したければ
プログラミングの
基礎を教えなさい

松林弘治
Kohji Matsubayashi

Teach your kids to code
to turn them into billionaires

「著作権保護コンテンツ」

あなたが
10年後に
生き残って
いるために

プログラミング
を知らない
ビジネスパーソン
のための
プログラミング
講座

A Programming Course for
Business People

福嶋紀仁 Norihito Fukushima

プログラミングができなくても、
「プログラミングはわかる」のが
一流のビジネスパーソンだ。

IT時代に不用品扱いされないための「必修科目」プログラミング。
これから始めるビジネスパーソンが最初に読んでおきたい1冊。

日経Kids+
日経ホームマガジン
子どもの未来を拓く!
[キッズプラス] PLUS

いよいよ
小中学校で
必修化!

親子で始める
プログラミング
[Programming]

親子で楽しく体験!
プログラムを
作ってみよう

文系でもよくわかる
そもそも
プログラムとは?

人気の小中高校生向け
プログラミングスクール

プログラミング必修化で
子どもたちは何を学ぶのか

親は
何をすべき?

Windows
パソコン
だけで
OK!

すごい! なぜ? どうして?
子どもといっしょに
コンピュータと
プログラミングを
学ぶ本

矢沢久雄 著

キー入力 | 計算処理 | 表計算 | 暗号化・復号 | 計算誤差
小数点数 | バグ・デバッグ | ゲーム | 画像処理 | 学習機能

日経BP社

親子で楽しみながら
考える力、つくる力、伝える力を育もう!

小学生から
はじめる
わくわく
プログラミング

Scratch 1.4/2.0
両対応

阿部 和広●著
アラン・ケイ●特別寄稿
ミッチェル・レズニック●序文
酒匂 寛●監訳

マサチューセッツ工科大学
MITメディアラボ レズニック教授 推薦!

これからの社会に不可欠な
3つの能力を身につけよう!
アラン・ケイ博士による「理想のコンピューターと教育」
創造力 論理的思考力 共創力

子供から
大人まで
楽しめる!

日経BP社

日経パソコン編
日経BPパソコンベストムック

この1冊で、子どもの
思考力、創造力、学力が
確実にアップ!

小中学生からはじめる
プログラミングの本
2018年版

監修 株式会社ドワンゴ
著者 松林弘治

プログラミング
は最強の
ビジネススキル
である

KADOKAWA

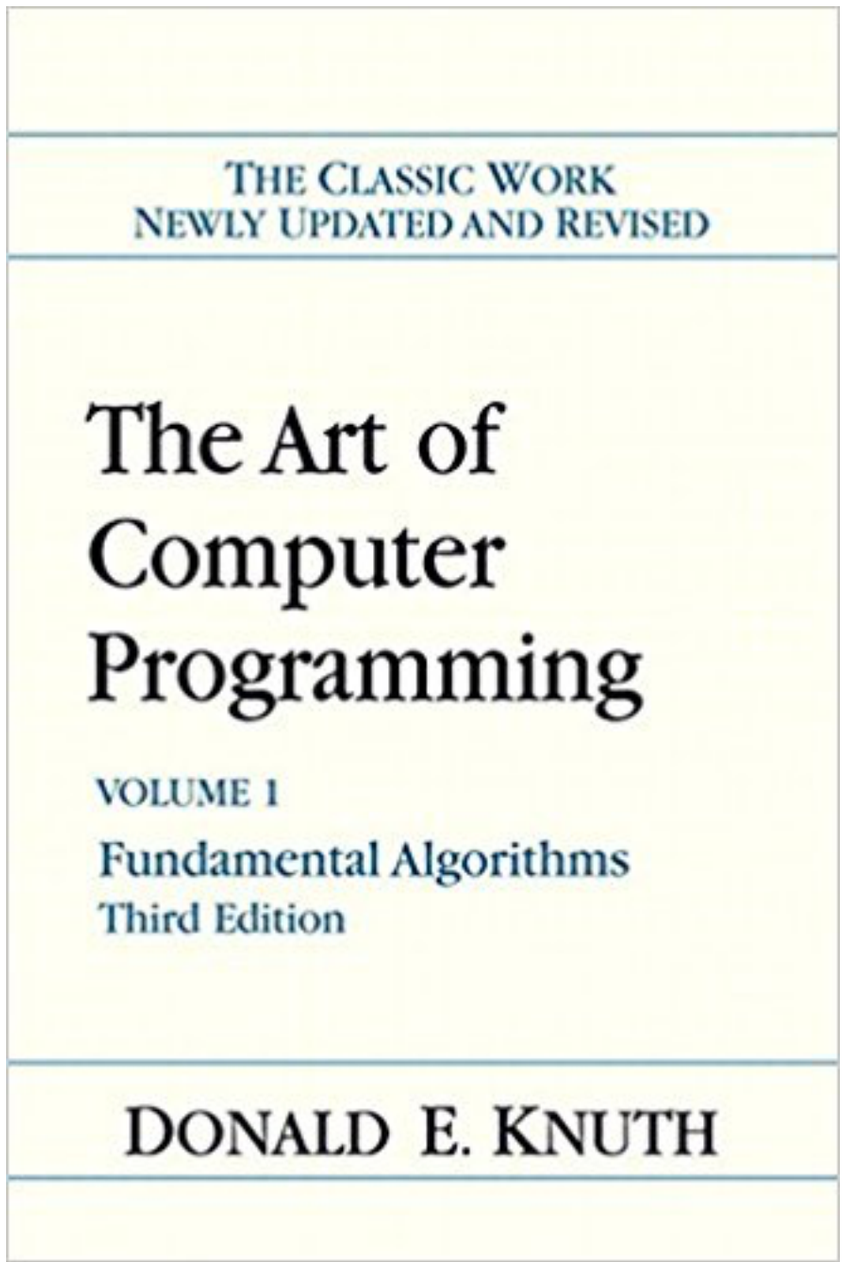
！？

2020年から始まるプログラミング教育では、

プログラミング言語を記述して、プログラムを動かすことは行いません (by 文科省)

プログラミングは「アート」by Don Knuth

バイブル 通称"TAOCP"



1974年ACMチューリング賞 受賞講演

Computer Programming as an Art

by Donald E. Knuth

When *Communications of the ACM* began publication in 1959, the members of ACM's Editorial Board made the following remark as they described the purposes of ACM's periodicals [2]: "If computer programming is to become an important part of computer research and development, a transition of programming from an art to a disciplined science must be effected." Such a goal has been a continually recurring theme during the ensuing years; for example, we read in 1970 of the "first steps toward transforming the art of programming into a science" [26]. Meanwhile we have actually succeeded in making our discipline a science, and in a remarkably simple way: merely by deciding to call it "computer science."

Implicit in these remarks is the notion that there is something undesirable about an area of human activity that is classified as an "art"; it has to be a Science before it has any real stature. On the other hand, I have been working for more than 12 years on a series of books called "The *Art* of Computer Programming." People frequently ask me why I picked such a title; and in fact some people apparently don't believe that I really did so, since I've seen at least one bibliographic reference to some books called "The *Act* of Computer Programming."

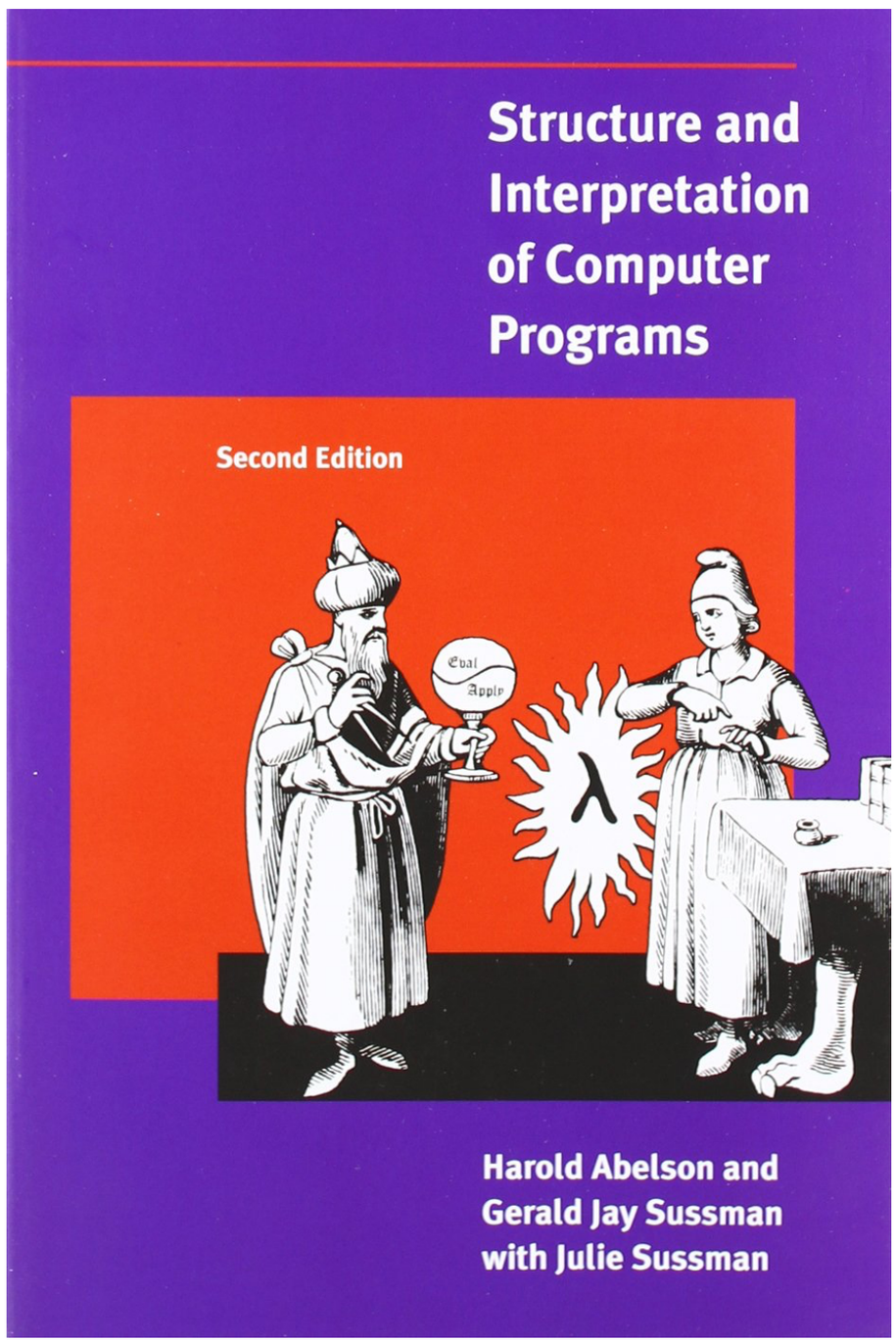
In this talk I shall try to explain why I think "Art" is the appropriate word. I will discuss what it means for something to be an art, in contrast to being a science; I will try to examine whether arts are good things or bad things; and I will try to show that a proper viewpoint of the subject will help us all to improve the quality of what we are now doing.

One of the first times I was ever asked about the title of my books was in 1966, during the last previous ACM national meeting held in Southern California. This was before any of the books were published, and I recall having lunch with a friend at the convention hotel. He knew how conceited I was, already at that

Author's address, Computer Science Department, Stanford University, Stanford, CA 94305

プログラミングは「魔法をかける呪文」

バイブル 通称"SICP"



We are about to study the idea of a *computational process*. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called *data*. The evolution of a process is directed by a pattern of rules called a *program*. People create programs to direct processes. **In effect, we conjure the spirits of the computer with our spells.**

我々の呪文でコンピュータの精霊を呼ぶ

"Why programming is a good medium for expressing poorly understood and sloppily-formulated ideas" (Marvin Minsky)

"人工知能の父"

WHY PROGRAMMING IS A GOOD MEDIUM FOR EXPRESSING POORLY UNDERSTOOD AND SLOPPILY-FORMULATED IDEAS

Marvin Minsky

MIT

This is a slightly revised version of a chapter published in *Design and Planning II -- Computers in Design and Communication*, (Martin Krampen and Peter Seitz, eds.), Visual Committee Books, Hastings House Publishers, New York, 1967.

There is a popular, widespread belief that computers can do only what they are programmed to do. This false belief is based on a confusion between form and content. A rigid grammar need not make for precision in describing processes. The programmer must be very precise in following the computer grammar, but the content he wants to be expressed remains free. The grammar is rigid because of the programmer who uses it, not because of the computer. The programmer does not even have to be exact in his own ideas-he may have a range of acceptable computer answers in mind and may be content if the computer's answers do not step out of this range. The programmer does not have to fixate the computer with particular processes. In a range of uncertainty he may ask the computer to generate new procedures, or he may recommend rules of selection and give the computer advice about which choices to make. Thus, computers do not have to be programmed with extremely clear and precise formulations of what is to be executed, or how to do it.

私とプログラミング

- 現在の専門は機械学習(人工知能技術),
  研究でプログラミングを日常的に使っている
- 大学に入るまでは特に経験はゼロ
  (父親のワープロをいじるのは好きだった)
- 情報工学を収める．学生時代にソフトウェア
  ベンチャーでプロジェクトリーダーも

私とプログラミング

- プログラミングでは何を作ってみるかが大事
- 英語話せること自体が大事なのではなくて，
  英語で「何をどう話せるか」が大事なのと一緒
- 当時も「情報化社会」とかで「パソコン」できるやつが将来有望とか言われていた
- 大学に入学した時，大学生協でパソコンを買った．マインスイーパに飽きた頃，プログラミングというのができることに気づく

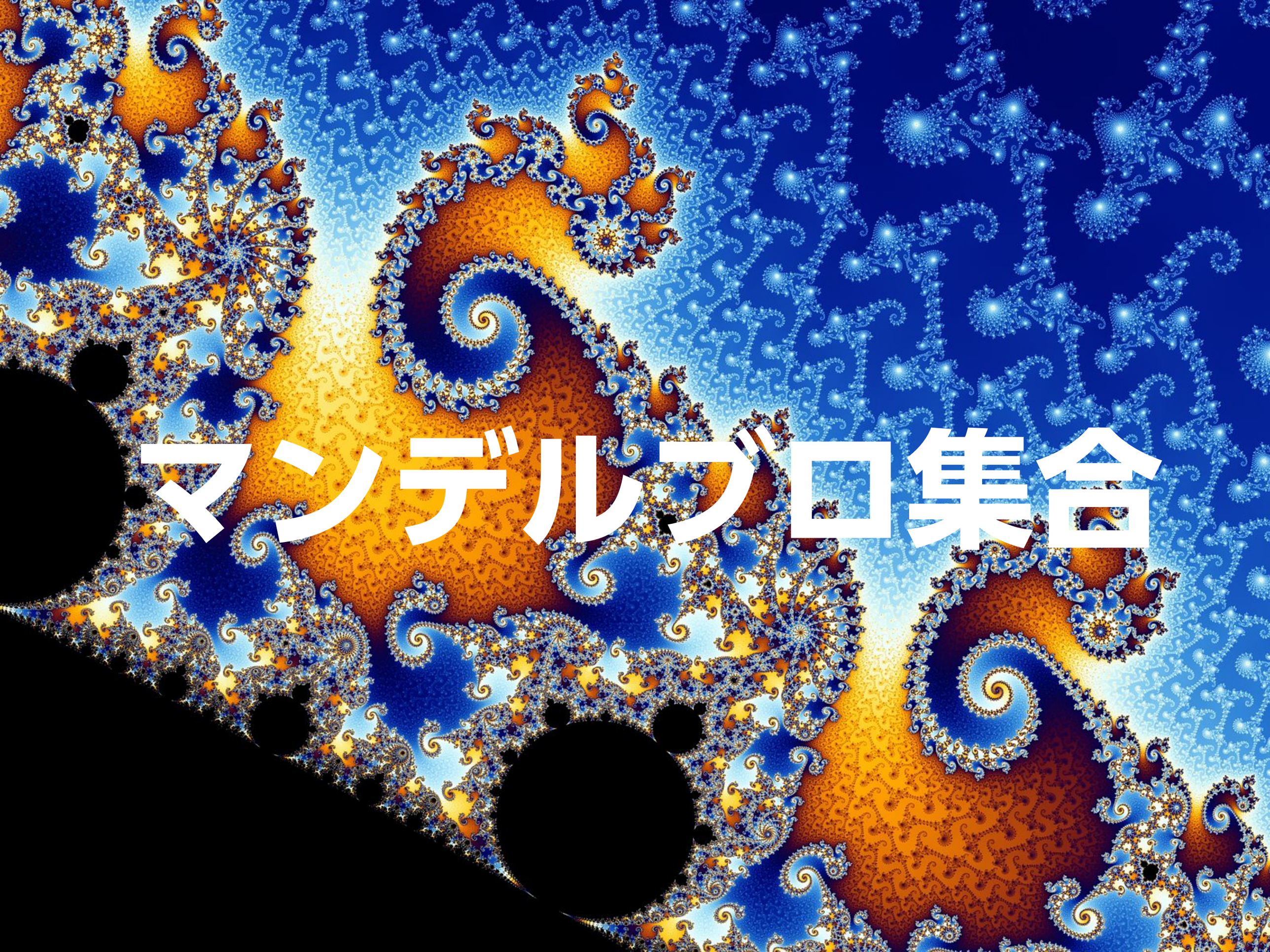
パソコンを手に入れてやってみたこと

- 音楽のプログラミング
  DTMやシンセサイザーのプログラミングに加え
  当時Common Lisp Musicという音響生成言語の本を見つけてちょっとハマった

- 情報処理技術者の資格の学校に行ってみた
  →資格勉強的に「COBOL」という言語を習う

- 大学の授業で「PASCAL」と「C」を少し習う

- お金を貯めて「DELPHI(Object Pacsal)」という商用パッケージを生協で購入, そしてどうしても作ってみたかったのが当時NHKスペシャルで知ったコレ！！


マンデルブロ集合

理屈は超絶シンプルだが出力が摩訶不思議で魅了される！

複素数列 $z_0 = 0, z_{n+1} = z_n^2 + c$ の発散の速さで色付け

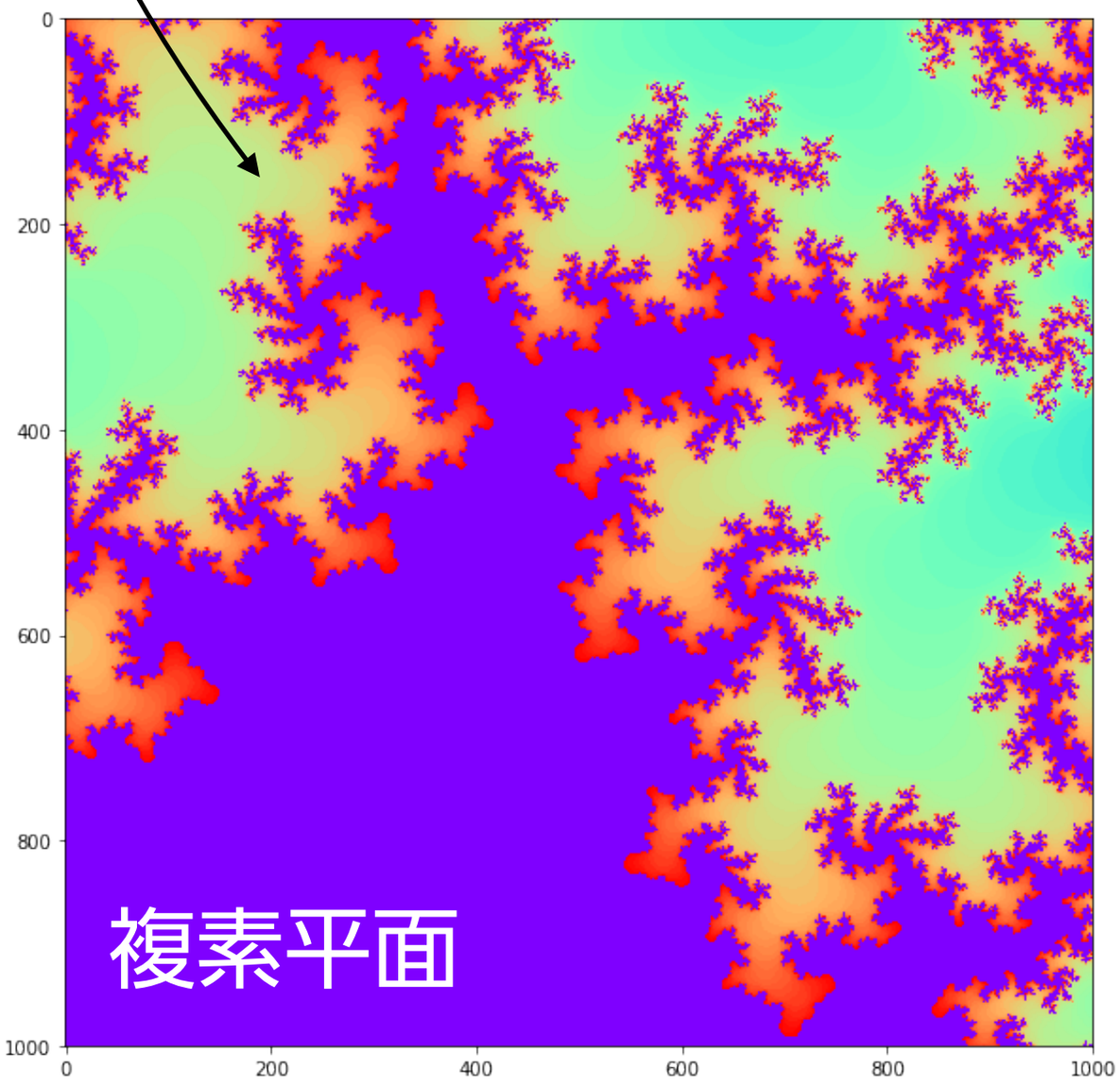
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

mpl.rcParams['figure.figsize'] = 10, 10
```

```
def mandelbrot(c, maxiter):
    z = c
    for n in range(maxiter):
        if abs(z) > 2:
            return n
        z = z*z + c
    return 0

def mandelbrot_set(xmin, xmax, ymin, ymax, w, h, maxiter):
    a = np.linspace(xmin, xmax, w)
    b = np.linspace(ymin, ymax, h)
    s = np.empty((w, h))
    for i in range(w):
        for j in range(h):
            s[i, j] = mandelbrot(a[i] + 1j*b[j], maxiter)
    return s
```

```
m = mandelbrot_set(-0.56, -0.55, -0.56, -0.55, 1000, 1000, 80)
plt.imshow(m, cmap='rainbow')
```



ちなみに DeepDream (by Google)

つまり下手だがプログラミングすると楽しかった！！

パソコンに入ってる写真ファイルを検索して，それを撮影した日付ごとに

2018_04_12などのフォルダ名を作り，そこへ格納して整理する (ruby)

```
# find . -name "P*.JPG" -print -exec cp {} /Volumes/Promise Pegasus/photobase/all/ \;
```

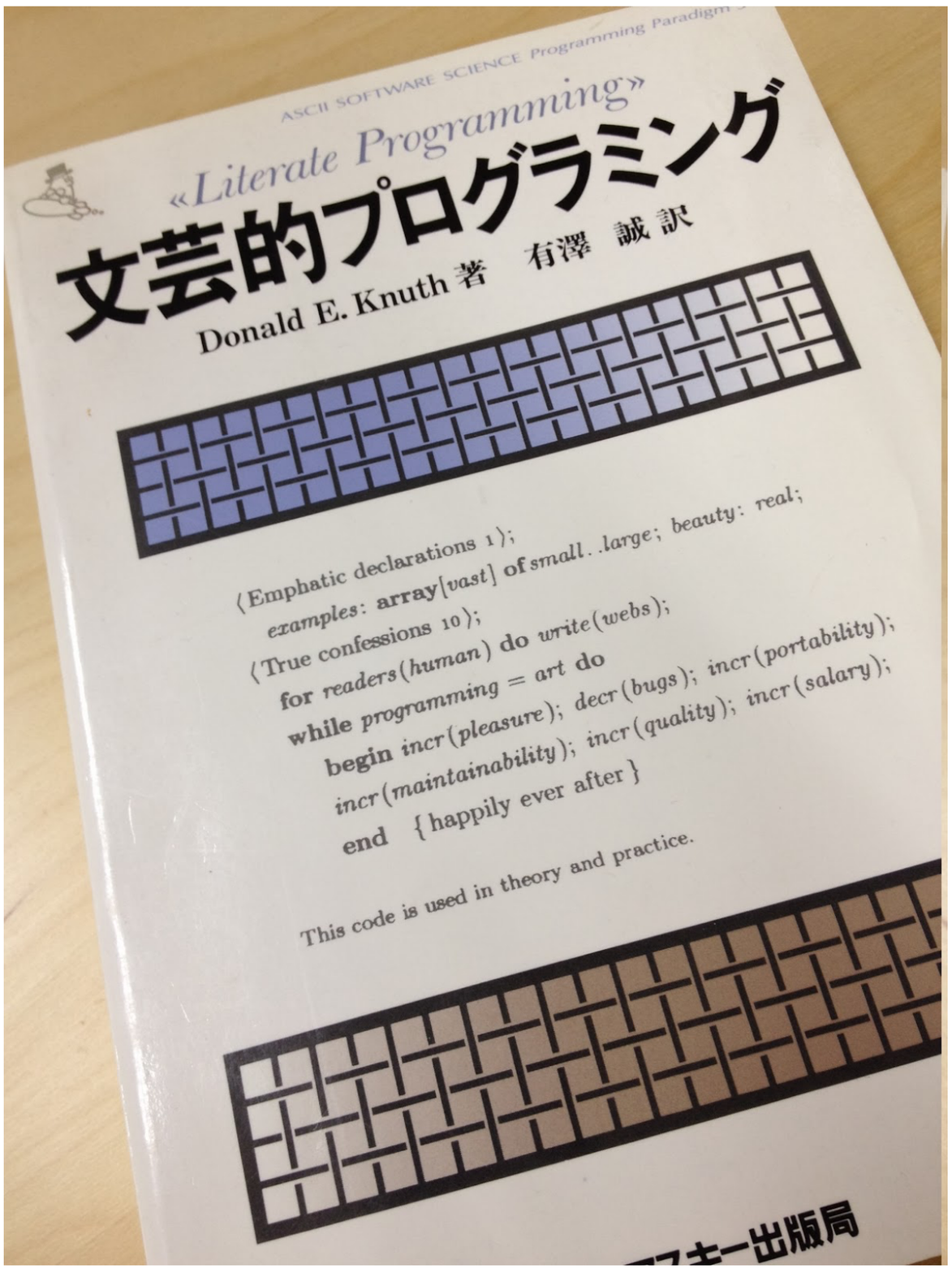
```
require 'exifr'

files = Dir.glob("/Users/takigawa/Desktop/our_photo/*.JPG")

puts "#{files.length} Photos"

files.each do |f|
  photo = EXIFR::JPEG::new(f)
  t = photo.date_time
  dirstr = "#{t.year}_#{t.mon}_#{t.day}"
  cmd = "mkdir -p /Users/takigawa/Desktop/photo_classified/#{dirstr}"
  system(cmd)
  cmd = "cp \"#{f}\" /Users/takigawa/Desktop/photo_classified/#{dirstr}"
  puts cmd
  system(cmd)
end
```

文芸的プログラミング



４年生の時図書館で見つけて面白かった先ほどのKnuthの本

(卒論はこれに沿って実装)

プログラミングとは芸術であり文学！！

Knuthの文芸的プログラミングは全く流行らなかったが...

「プログラムは人に読まれる」ために書くことは重視されるようになった！

- 昔は暗号のように分かりづらくても効率の良い職人芸が求められたが...
- 自分が書いたプログラムを自分でメンテするとは限らない
- 読みにくいとそもそも誤り(バグ)が入り込みやすいし，それが発見しづらい
- 最近の大規模ソフトウェアは複数人が関与するチームで作る

The Zen of Python (by Tim Peters)の抜粋いくつか

>> import this

Beautiful is better than ugly.

醜いより美しいほうがいい。

Explicit is better than implicit.

暗示するより明示するほうがいい。

Simple is better than complex.

複雑であるよりは平易であるほうがいい。

Readability counts.

読みやすいことは善である。

If the implementation is hard to explain, it's a bad idea.

コードの内容を説明するのが難しいのなら、それは悪い実装である。

美学がある！

オンラインで無料の良い講義も多数！

https://wiki.python.org/moin/BeginnersGuide/NonProgrammers

edX

Introduction to Python: Absolute Beginner

In this course that's perfect for true beginners, learn Python basics and start coding right away.

Microsoft

Introduction to Python: Fundamentals

Build on what you learned in the "Introduction to Python: Absolute Beginner" course, and dig into data structure basics.

Microsoft

UDACITY

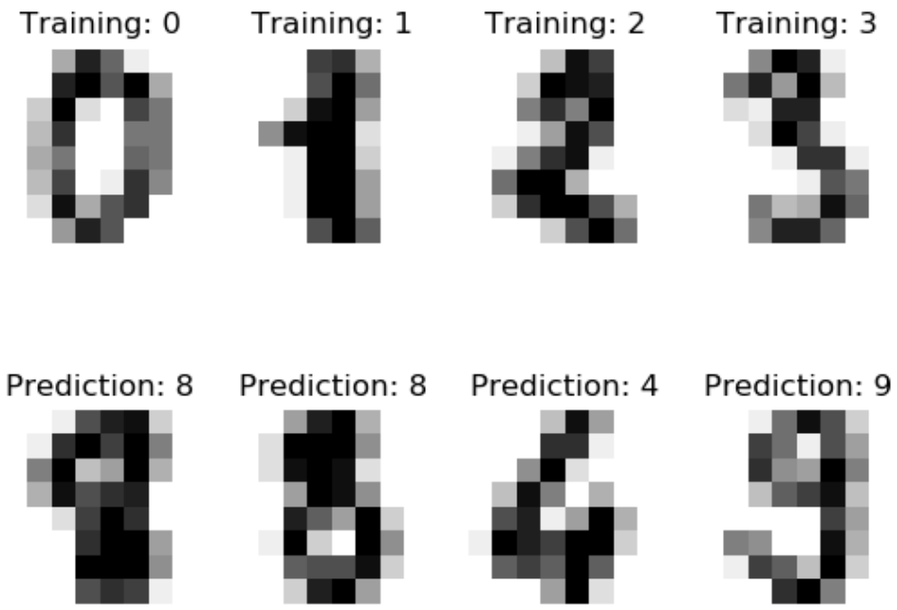
FREE COURSE

Introduction to Python

Starting Out in Python 3

START FREE COURSE

手書き文字認識も短く試せる！




```python
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics

digits = datasets.load_digits()

images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)

n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

classifier = svm.SVC(gamma=0.001)
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

expected = digits.target[n_samples // 2:]
predicted = classifier.predict(data[n_samples // 2:])

print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))

images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)

plt.show()
```

http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

心理的な障壁を感じる...という場合

Hour of Codeとは何ですか？

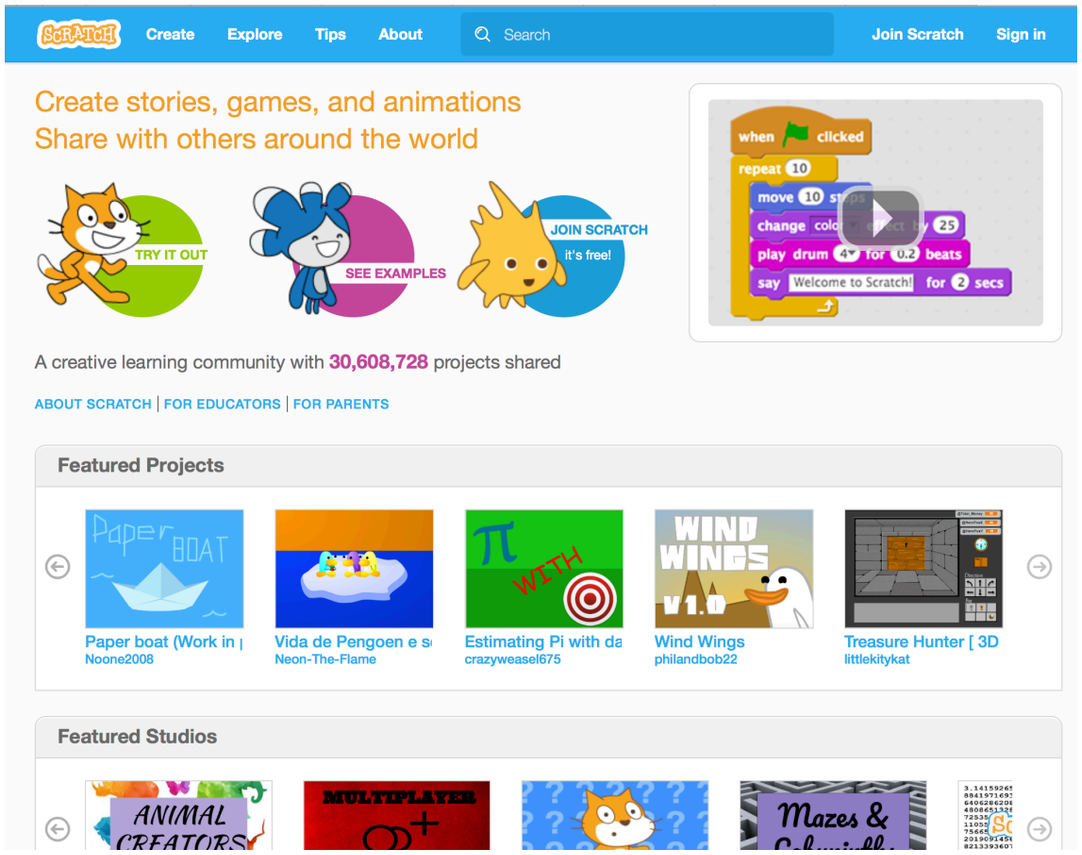
Hour of Code は、「コード（プログラミング）」の謎を解き明かし、全ての人が基本を学ぶことができることを示し、コンピューターサイエンス分野の幅広い参加者に向けて作成された、1時間のコンピューターサイエンス入門として始まりました。

まあ一種のゲームとして遊べます

「ビジュアル」プログラミング

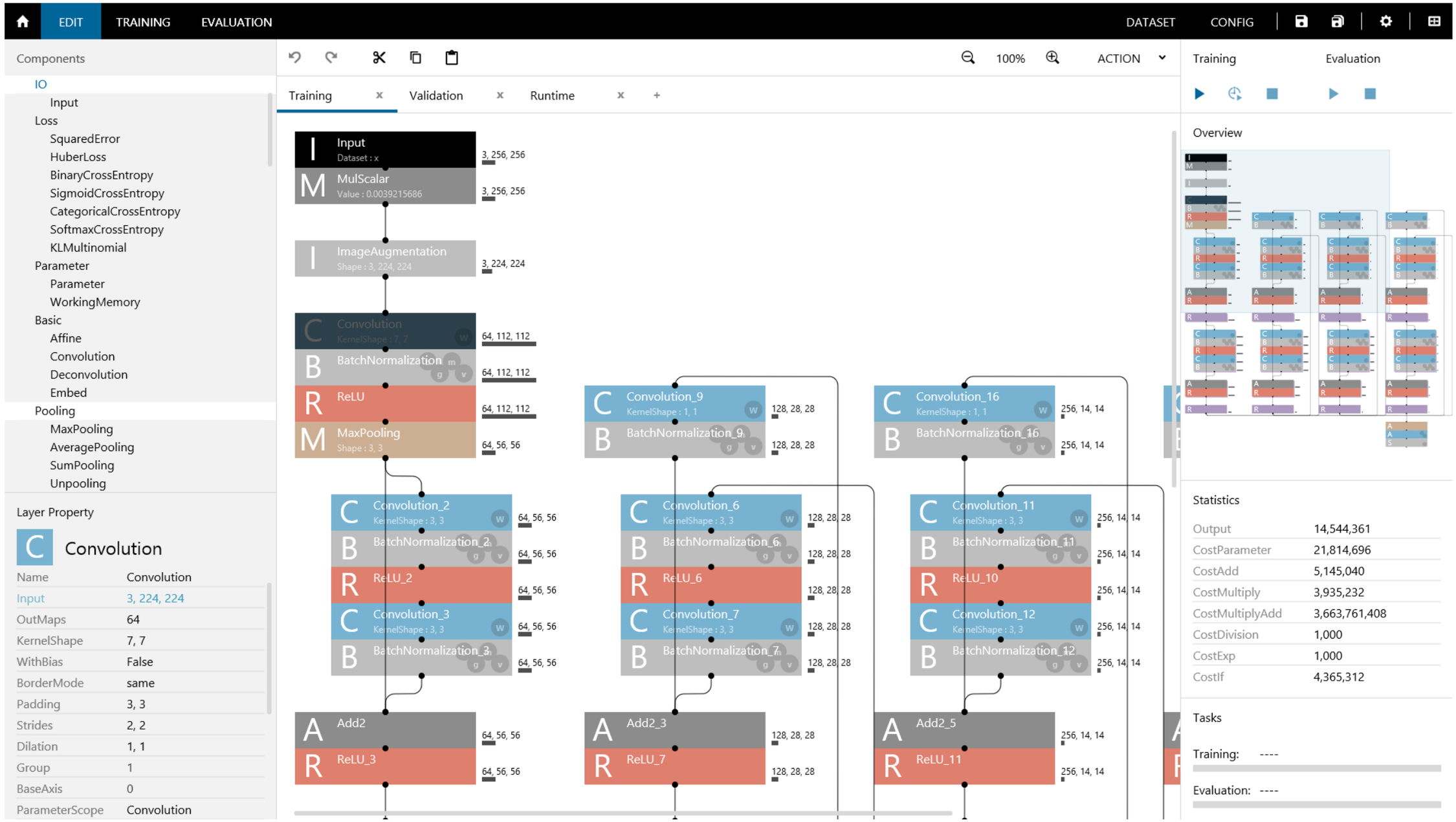
lightbot

Scratch



ちなみにdeep learningもvisualに

sony Neural Network Console



到達目標 (シラバスより)

1. 「プログラミングとは何か？」について自分なりの答えが持てるようになる。

2. プログラミングに最低限必要な「コンピュータの仕組み」や「アルゴリズム」「データ構造」と言った初歩的知識を身につける。

3. 実際にプログラミング言語を用いて簡単なプログラムを作成するための基本的なスキルを身につけ実践することができる。

4. 今後、プログラミングを自ら習得していくために何をしていけば良いかの理解や知識を得る。

5. コンピュータプログラムが我々の生活社会の基盤にあることを意識し、品質の高い美しいプログラムを作り上げることの価値を議論できるようになる。

授業計画 (シラバスより)

1. ガイダンス：授業の進め方を説明
2. プログラミングとは？(1)：自己紹介～コンピュータの仕組みとプログラム、その社会における役割
3. プログラミングとは？(2)：実習環境設定～初めてのプログラミング
4. 基本文法を学ぼう(1)
5. 基本文法を学ぼう(2)＋フォローアップ
6. 標準ライブラリを使ってみよう(1)
7. 標準ライブラリを使ってみよう(2)＋フォローアップ
8. プログラムの構造と設計(1)
9. プログラムの構造と設計(2)＋フォローアップ
10. データを読み解こう(1)
11. データを読み解こう(2)＋フォローアップ
12. 品質の高い美しいプログラムを作るには(「アート」な「呪文」への道)＋ディスカッション
13. 画像を認識するAIプログラムを作ってみよう(1)
14. 画像を認識するAIプログラムを作ってみよう(2)
15. 授業のまとめと振り返り：この後どうする？

予定

- 授業や演習への積極性(30%)
- レポートや成果物の内容(50%)
- ディスカッションへの貢献(20%)

授業では実際にプログラミングをしてみるとともに「分からない点，不思議な点，もっと掘り下げたい点」を適宜フィードバックしてもらい，その回答を次回授業に反映(フォローアップ・ディスカッション).
後半は各々の興味で自由制作や調査なども.

対象者

プログラミングに興味があるが、

(1)ほとんど、あるいは、全くしたことがない人

(2)してみたが理解がふわっとしていてよくわからなかった人

(3)したことがあるが自信がないので考え直したい人

(4)既にものすごくできるがなぜか急に入門に参加してみたくなった人

　(プログラミングを学ぼうとする同志をサポートしたい人)

(5)その他すべての人

次回について

- 教室が変わるので注意してください.

  4/19（木）・4/26（木）‥情報教育館Ｂ（情報教育館２階）

  5/10（木）～8/2（木）‥E209教室

- 8G以上の容量の「USBメモリ」を持参してください.