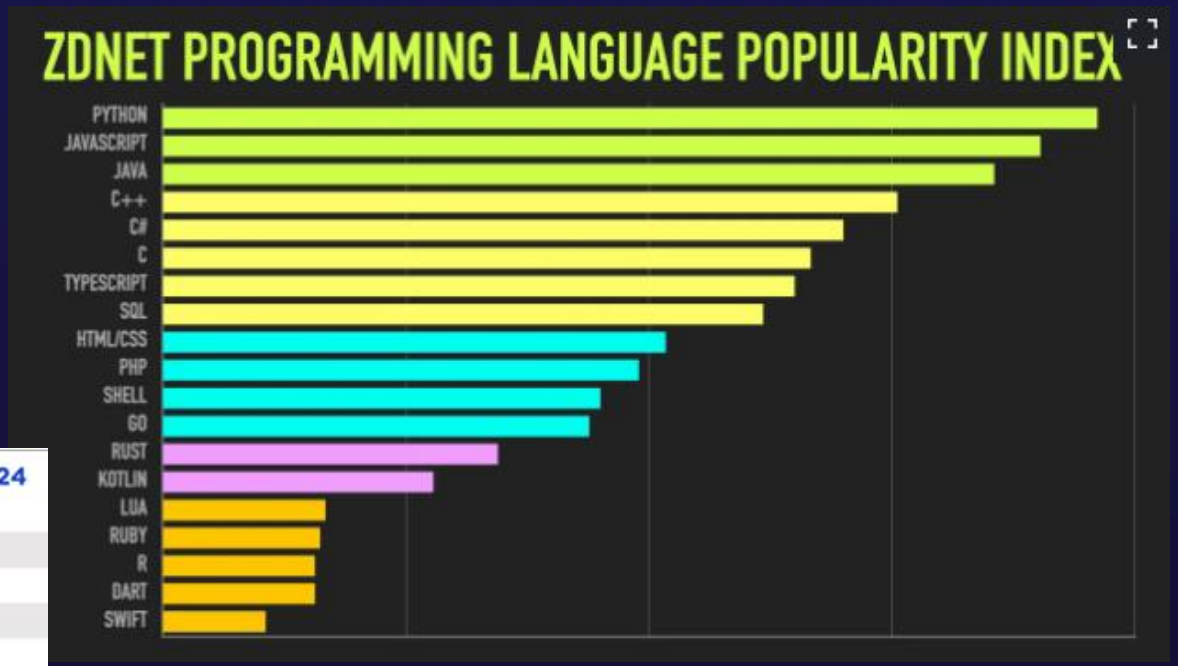
InterSystems
Creative data technology

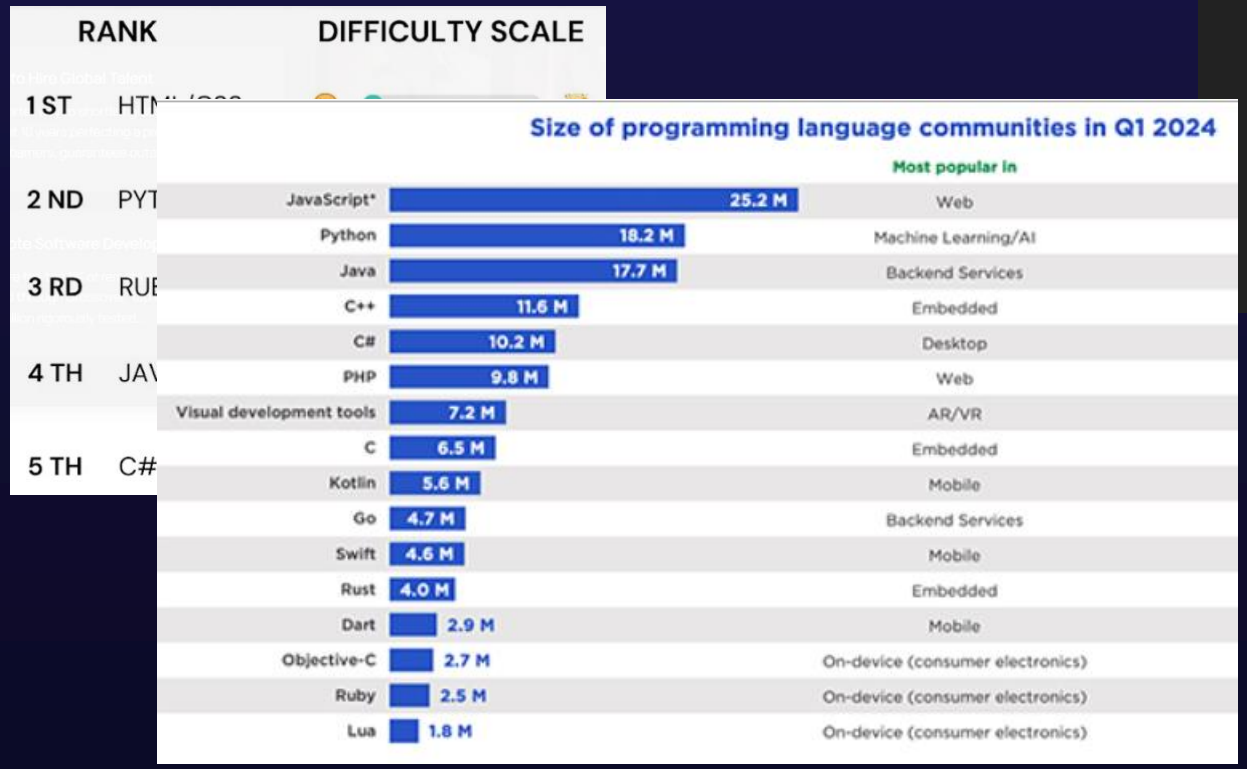
Python Workshop その1
Python入門

第2回 InterSystems Japan
開発者コミュニティミートアップ in 東京

Pythonとは

- 一番人気のプログラミング言語
- 分かりやすい
- 活発な開発者コミュニティ
- 豊富なライブラリー





出展

- The most popular programming languages in 2024 (and what that even means) | ZDNET
- 12 Programming Languages Ranked by Difficulty (+Chart)
- 59% of developers use AI tools & 25.2M JavaScript users

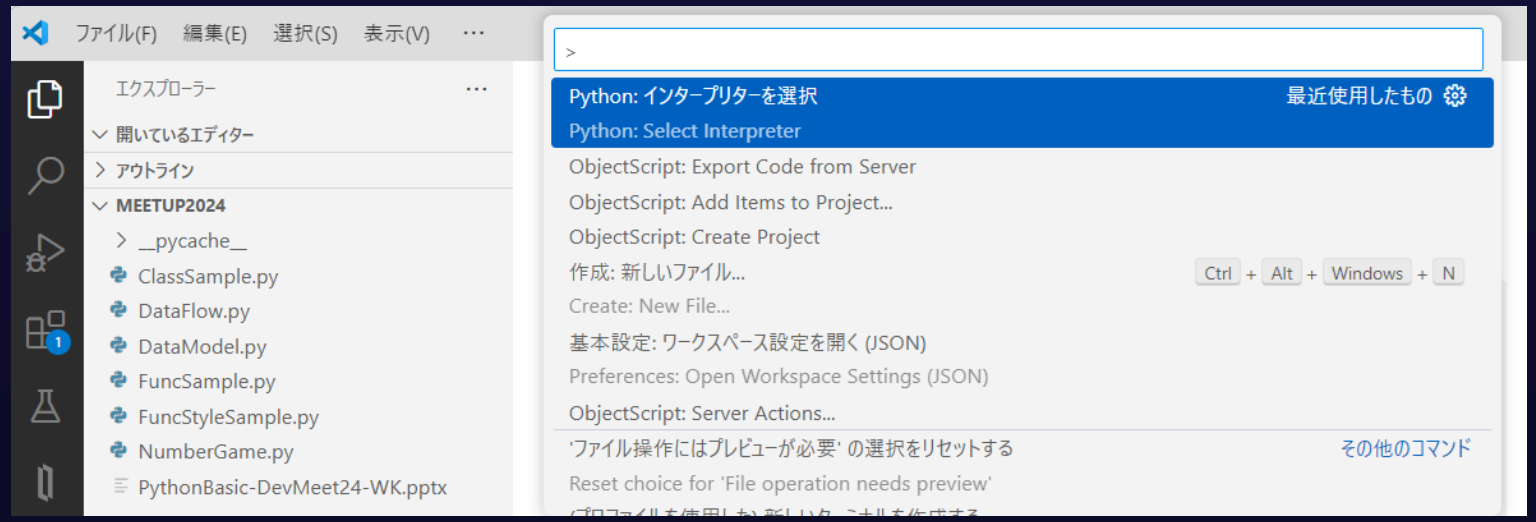
Pythonリファレンス 日本語版

[Python 3.12.7 Documentation]

環境設定について: Ready?

- コマンドプロンプトユーザー
  - IRIS 2024.1をインストールされている方
    - 環境変数PATHにC:¥InterSystems¥IRIS¥lib¥pythonを追加
  - 2024.2の場合
    - IRISに加えてPythonをインストール後、環境変数PATHに追加

- Visual Studio Code + Pythonエクステンション
  - Ctrl + Shift + Pキーでコマンドパレットを開き Python: Select Interpreterを入力、使用するPythonを選択



まずは動かしてみる

- インタラクティブモード

```
> python
Python 3.9.13 () [MSC v.1929 64 bit (AMD64)] on win32
Type “help”, “copyright”, “credits” or “license” for more information.
>>> 2+3*4
14
>>> print(“Hello IRIS!”)
Hello IRIS!
```

- Pythonスクリプトファイルの拡張子は.pyです

```
> python NumberGame.py
```

- 改行が文の区切り
- インデントで複合文を表現

サンプル: 数当てゲーム

```
# モジュールrandomをインポート（使用前に必要）
import random

# 正解を変数answerに代入
answer = random.randint(1,100)
in_str = input("1から100までの数を当てましょう")
# ユーザーが入力したデータを整数型に変換して変換guessに代入
guess = int(in_str)

# ユーザー入力と正解が不一致の間ループ
while guess != answer:
    if guess < answer:
        print("もっと大きいです")
    elif guess > answer:
        print("もっと小さいです")
    else:
        print("ここに来る?")
    in_str = input("1から100までの数を当てましょう、もう一度")
    guess = int(in_str)

print("お疲れ様でした、正解は %d でした" % answer)
```

データモデル - その1

- データは全てオブジェクト(object)として抽象化
- オブジェクトの特徴: 同一性(identity)、型(type)、値(value)
- オブジェクトの型でサポートする操作と取りうる値が決定
  - ✓組み込み関数 dir() でオブジェクト・モジュールの全属性を取得
- 値を変更できるタイプ（mutable)と変更不可のタイプ（immutable）
  - ✓Mutable: dict, list
  - ✓Immutable: 数値(int, float, complex), 文字列(str), tuple
  - ✓値が存在しない: None

- 変数にオブジェクトを代入

  myVar = 1234

  {変数} {=演算子} {オブジェクト}

データモデル - その2

- 数値型
  - ✓ boolはintのサブタイプ
  - ✓ 有理数（Fraction型）と10進浮動小数点演算（Decimal型）を標準ライブラリでサポート
- 文字列型
  - ✓ “（ダブルクォート)または’（シングルクォート)で囲みます
  - ✓ 文字コードはUTF-8
  - ✓ 特殊文字を含めるには¥（バックスラッシュ）で始まるエスケープが必要
  - ✓ “…”内の’、’…‘内の”はエスケープ不要
  - ✓ Rまたはrをクォートの前に付けるとraw文字列
  - ✓ Fまたはfをクォートの前に付けるとフォーマット文字列 、{}で区切られた式が展開されます

```
>m=11
>d=8
>print(f’今日は{m}月{d}日です’)
今日は11月8日です
```

データモデル - その3

- リストとタプル

  listSample = [1, 3, 5, 7]

  tupleSample = (2, 4, 6, 8)

  ✓ [n:m] n番目からm番目の前でを返す。負数の場合は後ろからカウント

  >print(listSample[1:3])

  [3, 5]

- 辞書型(dictionary)

  ✓ 不変のキーと値のペアでデータを管理

  - 複数要素のキーはlistでなくtupleを
  - 連想配列

  > emergencyTel = {‘Police’:110, ‘Ambulance’:119, ‘JCG’:118}

  > emergencyTel[‘Ambulance’]

  119

制御 – その1

```
x=5
if x > 3:
    print(“xは3より大きい”)
elif x == 3:
    print(“xは3”)
else:
    print(“xは2以下”)

a, b = 0, 1
while a < 10:
    print(a, end=“,”)
    a, b = b, a+b

for i in range(10):
    print(i, end=“,”)

for i in range(10):
    if ( i % 3 ) == 0 :
        continue
    print(i, end=“,”)
    if i == 8:
        break
```

- if … elif … else
  - ✓ 条件が真ならばブロックを実行
- while
  - ✓ 条件が真の間ブロックを繰り返し実行
- for … in
  - ✓ シーケンスの各要素(list, tuple, dictのキー, 文字列の各文字)に対して繰り返し処理
  - ✓ イテレーターから生成される各要素に対して繰り返し、詳細は後で
- ループの動きを変える文
  - ✓ continue, 次の項目処理に飛ぶ
  - ✓ break, ループから抜ける
- match
  - ✓ 他言語のswitch文に相当

制御 – その2

```
try:
    ans = 128/a
except ZeroDivisionError:
    print(“ゼロでは割れません”)
except Exception as e:
    print(f“その他エラー: {e}”)
else:
    print(“例外は発生しませんでした”)
finally:
    print(“ここで最終処理”)
```

```
with open( “test.txt”, ”r”) as f:
    line=f.read()
```

- try … except … else … finally
  - ✓ tryブロック実行中に例外が発生したら該当するexceptブロックへ遷移
  - ✓ 例外が発生しなければelseへ
  - ✓ 例外の有無に関わらず最後はfinallyへ
  - ✓ try外での例外発生に注意

- with
  - ✓ ブロックを抜ける時にオブジェクトの終了処理を呼び出す

関数

```
def seriesSum(a, b):
    num = b – a + 1
    wa = a + b
    ans = num * wa / 2
    return ans
```

- defの後ろに関数名と引数を括弧で囲んだリストで定義
- 呼び出す際には関数名を使用
- デフォルト引数、位置引数、キーワード引数

```
def hakobu(place, item='箱', num=5):

hakobu('倉庫')
hakobu('倉庫', 'ケース')
hakobu('倉庫', num=10)
```

```
> a = lambda x : x + 1
> a(10)
11
>a(a(a(10)))
```

- lambda式
  - ✓ 関数に引数として渡せる関数

クラス

```
class Animal:
    spec = ‘Mammal’              #クラス変数

    def __init__(self, name): #コンストラクタ
        self.name = name #インスタンス変数

    def getName(self):          #アクセサ
        return self.name

class Cat(Animal):              # 継承
    spec = ‘Cat’

import dataclasses
@dataclasses.dataclass
class DataclassPerson:
    number: int
    name: str

person = DataclassPerson(1234,”iris taro”)
print(person.name)
```

- データと機能を組み合わせる方法-> クラス
- クラスメンバは全てpublic
- メンバ関数は全てvirtual
- メンバ関数の第1引数はオブジェクト自体

- データクラス
  - ✓ C言語での構造体相当
  - ✓ データをまとめるだけならばデータクラス
  - ✓ メリットは__eq__関数が自動生成される等

- デコレータ
  - ✓ 関数、クラス自体のコード変更は無く、前後に処理を追加
  - ✓ @dataclasses.dataclassはビルトインデコレータのひとつ
  - ✓ @property, @classmethod, etc
  - ✓ 主なデコレータは functools モジュールで定義

モジュール・パッケージ

- 誰かが書いた便利な機能を使いたい時に
- 定義をファイルに書いておき、インタープリターやスクリプトファイルで読み込む

＝> Pythonの定義や文が入ったファイルをモジュール(module)と呼びます

- パッケージ
  - ✓ 複数のモジュールを構造化する手段
  - ✓ __init__.pyを持つフォルダと.pyファイル

- モジュールは取り込み(import)して使用
  - ✓ from パッケージ import モジュール

  >from math import pow

- モジュールがimportされたか、直接実行されたか
  - ✓ Pythonコマンドから直接実行の場合、変数__main__の値が”__main__”になります

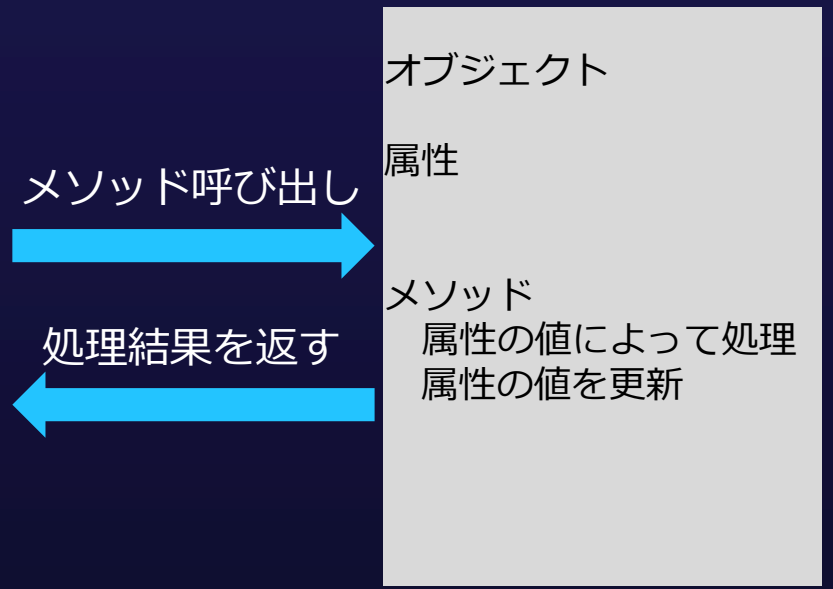
```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
```

```
if __name__ == '__main__':
    sys.exit(main())  # next section explains the use of sys.exit
```

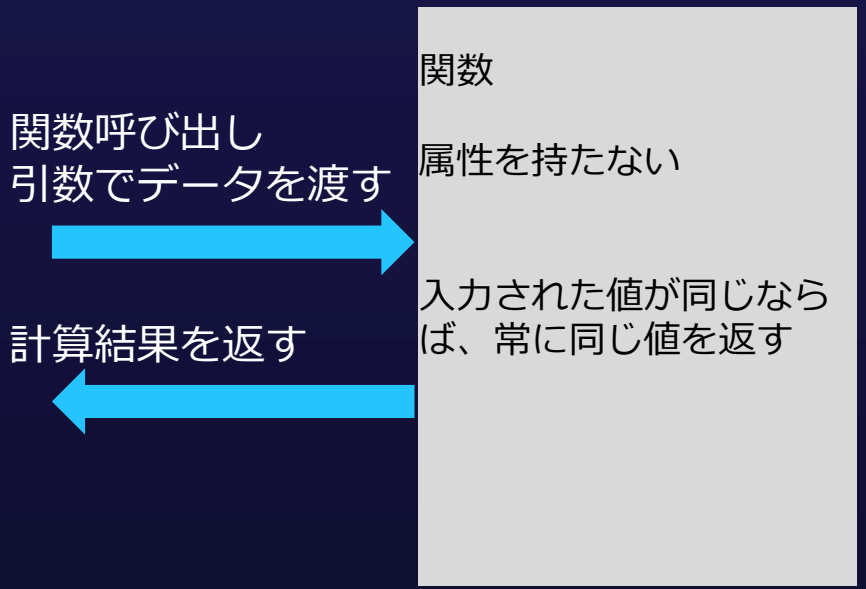
関数型プログラミング

- 手続き型
  - ✓ 殆どのプログラミング言語
- 宣言型
  - ✓ SQL
  - ✓ 問題を説明だけして、実際の処理内容は内部で決定
- オブジェクト指向
  - ✓ オブジェクトには内部状態があり、内部状態に関わる・変えるメソッド
- 関数型
  - ✓ 関数は入力を受けて出力
  - ✓ 同じ入力なら同じ出力になるよう、内部状態を持たない
  - ✓ 内部状態を変えたり、返り値に現れない変更のような副作用は好ましくない
    - ユーザー入力・画面に出力等
  - ✓ 問題を細かく分割しやすく、テストやデバッグしやすい
  - ✓ 再利用性が高い

オブジェクト指向型と関数型の比較



オブジェクト指向



関数型

イテレータとジェネレータ

```
> def generate_ints(N):
    for i in range(N):
        yield i

> gen = generate_ints(2)
> next(gen)
0
> next(gen)
1
> next(gen)
StopIteration
```

- イテレータ
  - ✓ 1度に1つの要素ずつデータを返すオブジェクト
  - ✓ 要素が無い場合はStopIteration例外発生
  - > for x in Y
    - ✓ Yはイテレータかiter()でイテレータを作れる
  - ✓ 全要素に対して操作、フィルタリング等
- ジェネレータ
  - ✓ イテレータを書く作業を簡単にする特殊な関数
  - ✓ ローカル変数が保存され、処理を続行できる
  - ✓ 巨大なデータセットを計算する時に有効

ジェネレータ式とリスト内包表記

```
> listSample=['InterSystems', 'iris']
> upIter = ( li.upper() for li in listSample )
> for item in upIter:
>     print(item)

INTERSYSTEMS
IRIS

> upList = [ li.upper() for li in listSample ]
> upList
['INTERSYSTEMS', 'IRIS' ]

> lgList = [ i for i in listSample if len(i) > 4 ]
> lgList
['InterSystems' ]
```

- ジェネレータ式
  - ✓ イテレータの出力に対してデータ変換やフィルタリングを行い新しいイテレータを返す
- リスト内包表記
  - ✓ イテレータの出力に対してデータ変換やフィルタリングを行い新しいリストを返す

イテレータとよく一緒に使われるビルトイン関数

```
> listSample=[‘InterSystems’, ‘iris’]

> upp=lambda s: s.upper() #大文字にする
> list(map(upp, listSample))
['INTERSYSTEMS', 'IRIS’]

> pred=lambda s : 1 if len(s) > 4 else 0
> # 5文字以上ならTrueを返す
> list(filter(pred, listSample))
[‘InterSystems’]

> for x in enumerate(listSample):
>     print(x)
(0, 'InterSystems')
(1, 'iris')
```

- map
  - map(f, iterA, iterB, ..)は以下のイテレータを返します f(iterA[0], iterB[0]), f(iterA[1], iterB[1]),
- filter
  - filter(predic, iter)はiterからpredicを満たすイテレータを返します
- enumerate
  - enumeate(iter, start=0)はイテラブルの要素に番号を振り、要素を含むタプルを返します

効率的なループのイテレータ生成関数: itertools

```
>import itertools
>srclist=[1,3,5,7,9,11]

# 条件がfalseの要素を抽出
>ta = itertools.filterfalse( lambda x :  x % 3  == 0, srclist)
>list(ta)
 [1, 5, 7, 11]

# 条件がtrueの間は要素を返し、一度falseになると終了
>tb = itertools.takewhile( lambda x : x < 8, srclist)
>list(tb)
[1, 3, 5, 7]

# 条件がtrueの間は何も返せず、一度falseになると残りを全て出力
>tc = itertools.dropwhile( lambda x : x < 8, srclist)
>list(tc)
[9, 11]
```

最新のPython

- GIL（Global Interpreter Lock）
  - ✓ 内部にあるこのロックを使って、Pythonでは一度に一つだけのスレッドが実行されることを保証
  - ✓ スレッド間の切り替えをバイトコード命令の間でのみ行う
  - ✓ Numpy等のC言語で書かれたライブラリーは別
  - ✓ GILを無効にする実行ファイルが実験的な機能として3.13で追加

- JIT（Just-In-time）コンパイラ
  - ✓ 現状は数%の速度向上
  - ✓ 3.13では専用のオプションを指定してビルド

最後に

Thank Python official document, 3.12.7 Documentation, for all information given.

Have a good coding time with Python, today!

InterSystems®
Creative data technology