コマンドラインインタフェース上でのPython実行/標準入出力

名古屋大学 情報基盤センター
情報基盤ネットワーク研究部門
基盤ネットワーク研究グループ
嶋田 創

概要

- Pythonを対話型インタフェースで実行してみる
- Pythonプログラム(スクリプト)の実行
- 基本的な変数
- 標準入出力
  - print関数、input関数
  - 入出力に関するTips

Pythonの対話型インタフェース

- シェルより「python3」と入力すると開始
  - 前回課題でちゃんとalias設定すれば、「py」の短縮呼び出しも可能
- プログラムを1行1行入力して実行することが可能
  - 「この書式OK?」「ライブラリあるかな?(import可能?)」な確認に便利
- 式はそのまま評価される

```
[shimada@ssh ~]$ py
Python 3.5.3 (default, Mar 14 2017, 16:27:17)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-17)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world")
Hello world
>>> 3+5
8
>>> a=3
>>> b=5
>>> a+b
8
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
```

対話型インタフェースのTips

- 終了はCtrl + D
  - 参考: シェル上でのプログラムの強制停止はCtrl + C(他言語でも)
- bashと同様に矢印キーなどで履歴も含めて編集可能
- 変数を利用したり、ライブラリ(後述)をimportして利用も可能
- 数式では括弧は小括弧のみを使う
  - 小括弧のみで階層を考えて式を記述
- 変数に文字(列)を代入する場合は"や'でくくる(例: a = "xyz")
  - 使わない方のクォートは文字列に含めることができる
    - ¥'や¥"のようにエスケープしてもOK
  - """や'''でくくれば改行まで文字列に含めることができる

Pythonで使える基本的な演算子

- 加算: a + b
  - 文字列を加算すると連結される
- 減算: a - b
  - 変数に単にマイナスをつけて正負反転も可能
- 乗算: a * b
  - 文字列と整数の乗算で文字列の繰り返し
- 除算: a / b
- 除算(小数点以下切り捨て): a // b
  - 負の数の切り捨ては「小さい値になる側に切り捨て」という点に注意
- 剰余: a % b
- 累乗: a ** b

基本的な変数の型

注: 以下、プログラムの表記中のイタリック体は「任意の変数が入る」ことを示します

- 文字列型(string)
  - 文字列型に変換: *格納先変数* = str(*文字列型に変更する変数*)
- 整数型(integer)
  - 整数型に変換: *格納先変数* = int(*整数型に変更する変数*)
- 浮動小数点型(floating point、FPと略すことも)
  - 小数点がある数
  - FPに変換: *格納先変数* = float(*浮動小数点型に変更する変数*)

- アルファベット、数字(2文字目以降)、_が変数として利用可
  - Pythonの記述で使われる単語(予約語)は使用不可
- 型の違う変数を混ぜて操作しようとエラーが出る点に注意
  - 例: a="abc", b=3として、a+bを実行すると怒られます

Pythonスクリプトを書く

- VS codeで新規作成→Pythonプログラム→ファイル名指定
  - 末尾に.pyがついたファイルが生成される
  - Pythonモードを持つエディタなら、末尾.pyのファイルでモードに入る
- 最初の行で「#!/usr/bin/env python3」とPythonインタプリタを指定
  - 次スライド2.のやり方で実行するなら不要だが、支障は無いのでおまじないのつもりで書いておく
- "#"の文字より行末まではプログラムとして解釈されないコメントとなる(インタプリタの指定を除いて)

例: hello.py

```
#!/usr/bin/env python3

# お約束のHello world。worldとは「その言語の世界」という噂。
print("Hello world")
```

Pythonスクリプトを実行

1. シェルから実行属性をつけたPythonインタプリタを呼び出す
    1. $ chmod +x hello.py
        - 実行属性(eXecution)を付与(+)
        - IDEなどの設定次第では、自動的に実行属性を付与してくれる
    2. $ ./hello.py
        - LinuxはWindowsと違って、今いるディレクトリ(カレントディレクトリ)に実行可能ファイルがあっても、プログラム名だけでは実行できない
        - ディレクトリレベルから実行ファイルを指定することで実行可能(「./」は現在のディレクトリを指す)
        - カレントディレクトリにパスを通せばプログラム名だけで実行可能だが、安全のためにおすすめしない(変なプログラムの誤実行の元)
        - ちゃんと実行属性がついていないと「bash: ./hello.py: 許可がありません」と言われる
2. シェルよりPythonインタプリタとスクリプトファイルを指定
    1. $ python3 hello.py

エラーが出た時には

エラーは以下の形で表示される

- 「どのファイルのどのモジュールの何行目 + 行の内容の表示」の形でエラー箇所が表示される
  - (まだ複数ファイルやモジュールは取り扱っていないが…)
- エラーの内容が英語で表示される
  - **ちゃんと英語読むこと!**
  - 下図では「prontは(関数として)定義されていない(名前のエラー)」と出ている
    - 文字の打ち間違い(typo)は一番多いエラーの原因

```
[shimada@ssh ~]$ ./error.py
Traceback (most recent call last):
  File "./error.py", line 3, in <module>
    pront("Hello World!")
NameError: name 'pront' is not defined
[shimada@ssh ~]$
```

よくあるエラーとそのよくある原因(1/2)

- SyntaxError: invalid syntax
  - 「文脈がおかしい」というエラー
  - typoでよく発生する
- SyntaxError: unexpected EOF while parsing
  - 「プログラムを解釈している途中に突然終わった」というエラー
    - EOFはEnd Of Fileの略
  - 作りかけだったり、行が抜けてたり(間違って複数行削除)とか
- SyntaxError: Missing parentheses in call to 'print'
  - print関数の呼び出し時に括弧が抜けている(Python 2の書式)
- NameError: name '*変数名*' is not defined
  - 定義されていない変数を利用した時に発生
  - 「予約語を間違える → 間違えた予約語を変数と解釈 →無い」でよく発生

よくあるエラーとそのよくある原因(2/2)

- IndentationError: expected an indented block
  - インデントされたプログラムのブロックがあるはずなのに無い
  - 作りかけだったり、行が抜けてたり(間違って複数行削除)とか
- TypeError: unsupported operand type(s) for +: 'int' and 'str'
  - 整数型(int)と文字列型(str)を+で加算しようとした
  - ちゃんと型変換(スライド15や次回のスライド参照)する
- ImportError: No module named '*モジュール名*'
  - 指定したモジュールが存在しない
  - typoだったり、システムにモジュール(ライブラリ)が入っていなかったり

プログラムに関するTips(1/2)

- 変数は英語を基本とする
  - 基本的に中身が分かりやすい表記とする
    - 例: box_height, BoxWidth (自分でコーディングルールを作る)
  - 長い単語は略すのもあり(例: variable → var)
    - プログラム中で略し方がバラつくのはだめ (コーディングルール作る)
  - 予約語は変数として使えない
    - VS Codeが他の変数と違う色でハイライトしたら多分予約語なので注意
- コメントは効果的に使おう
  - 「自分がこのプログラムを忘れた頃に見直して、ここが理解しにくいだろうな」という所に理解を助けるコメントを入れるのが基本
    - 一目瞭然の物は書かない(例:「# aに10を代入」)
    - プログラムの記述の説明ではなく、処理(やること、やりたいこと)の説明を入れる (例:「# ここから入力値のチェックと再入力依頼」)
  - 「# ここ後で直す」という自分へのコメントを臨時で入れるのもあり

プログラムに関するTips(2/2)

- 定数の利用
  - 後から変更される可能性のある数字をプログラムに埋め込むのは、修正時に変更漏れがあったりして好ましくない
    - プログラム記述者以外が読むと「なんでこの数字が出てくるのか」が非常に理解しにくいため、「マジックナンバー」と呼ばれる
  - 例: 学籍番号を4で割った剰余で4グループにグループ分けするプログラム
    - 途中から6グループに分けることになったら?
    - 定数「NUM_GROUP = 4」を定義してそれで剰余を取っておいてあれば、定数宣言部のみを修正すればOK
  - Pythonには定数型が無いので、「特定の変数を定数として用いる(途中で変化させない)」という暗黙のルールで運用する
    - 大文字のみで書かれた変数は定数扱いするという暗黙のルールがある

変数の型などを調べたい時

以下のコマンドが便利

- type(*変数名*): 変数の型を返す
  - 例: type(4.5)　→<class 'float'>
- dir(*変数名*): その変数に対して適用可能な演算やメソッド(後述)を一覧表示
- help(*変数名*): その変数に対して適用可能な演算やメソッド(後述)を説明付きで一覧表示
  - dirの上位互換

標準入出力

- プログラムとして実行時に重要
  - 対話型インタフェースで使っても問題なく動く
- 標準出力: シェルへの文字出力
  - print関数で実現
    - 書式: print("文字列")
    - 書式: print(*文字列を格納した変数*)
- 標準入力: シェルからの文字入力
  - input関数で実現
    - 書式: *入力を格納する変数* = input("提示文字列")
      - 例: name = input("名前は? ")
    - 変数は文字列型になる点に注意 →int(*変数*), float(*変数*)で変換
- print関数、input関数ともにPython3から書式が変更されているので、古い資料を参照する時に注意

print関数/input関数のTips

- print関数で表示する変数(結果)に対する装飾はよくやる
  - 例: 変数totalに合計金額が入っている
    - 装飾無し例: print(total)
    - 装飾例: print("合計金額は" + str(total) + "円です。")
  - 文字列と数字を混ぜる時には文字列に統一する点に注意
  - print関数の外で装飾して別変数に代入しておいてもOK
  - より詳細な装飾はformat関数を利用
    - 左右中寄せ、数字の桁数指定(による揃え)、空白調整などが可能
- input関数の入力値は文字列になることに注意
  - 必要に応じてint(*変数*)やfloat(*変数*)で整数型や浮動小数点型に変換
- 複数行の文字を提示したい場合
  - エスケープシーケンスで改行文字を入れる(次スライド)
  - """や'''でくくって文字列中で改行可能とする

エスケープシーケンスによる特殊文字

- エスケープシーケンスとは特殊な文字の表現方法
  - 例:「"」を表示したいけどPythonが文字列の終了の"扱いしてしまう…
- ¥(Linuxでは"＼"で表示されることも)の後に規則に従って文字を入力
- 代表例
  - ¥(改行) : 改行を無視(次の行とつながる)
    - 「行末に¥を入れると改行が無視されて次の行とつながる」感じ
  - ¥¥ : ¥という文字
  - ¥' : 'という文字
  - ¥" : "という文字
  - ¥t : タブ文字(x方向に8の倍数の位置まで空白が入る)
  - ¥r¥n : Windowsの改行
  - ¥n : Linux/MacOSの改行