オブジェクト指向プログラミング

第１１回

箕原辰夫

文字列の入力

- 文字端末から
  - 文字列変数 = input("プロンプト文字列")
  - s = input("単語入力")
  - s = input("一行入力")
  - s = input( )　# プロンプト文字列は省略可能

tkinterでの文字列入力

- Entry… 1 行だけ、リターン（Enter）キーで、入力終了となる
  - 例：entry = Entry(window)
  - insert(Tkinter.END,"追加する文字列") で、デフォルトの入力文字を指定しておける
    例：entry. insert(Tkinter.END,"Apple and Orange")
  - Entryオブジェクト作成時に、「width=半角文字の文字数」で大体の幅を指定できる
    例：entry = Entry(window, width = 50)
  - get関数で入力された文字列を取り出すことが出来る
    例：text = entry.get( )

tkinterで複数行の文字列入力

- Textクラス、あるいはtkinter.scrolledtextモジュールのScrolledTextクラスを使う
  - Text(window, オプションのパラメータ)
  - **from** tkinter.scrolledtext **import** * # ScrolledTextを使う場合
  - ScrolledText(window, オプションのパラメータ)
  - 例：ScrolledText(win, width=50, height=25, font=("Times Roman", 20))
- Textクラス、ScrolledTextクラスでは、中の文字列を設定するのに、insertとdeleteメソッドを使う
  - 例：sctext.delete(1.0, "end"); sctext.insert(1.0, wholelines) # 1.0は先頭を示す
- Textクラスの詳細は、以下のページから
  - https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html

tkinterでラベルで文字列表示

- ラベル表示後に、表示文字列の内容を取り替えたい場合は、StringVarクラスのオブジェクトを用意して、textキーワード引数ではなく、textvariableキーワード引数で、そのオブジェクトを指定する。
- 設定例：

```
entrytext = StringVar( )
result = Label( window, textvariable=entrytext, width=40 )
result.pack( )
```

- コールバック関数などで、StringVar()オブジェクトのset関数で、内容を取り替える

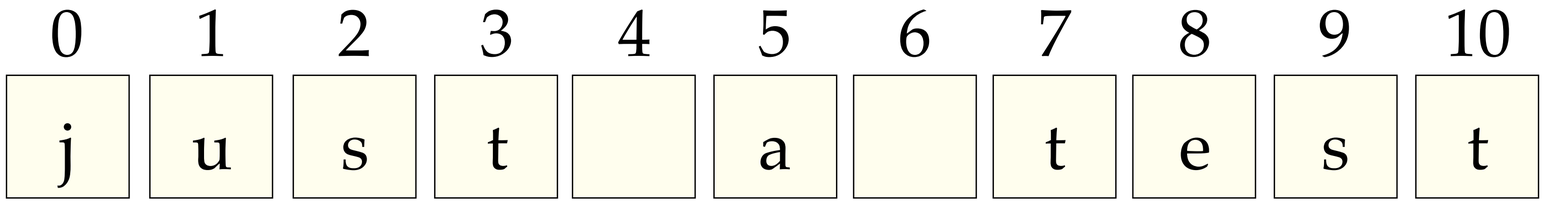
```
text = entry.get( )
entrytext.set( "入力されたのは：" + text )
```

文字列の比較、照合など

- **文字列は、==で等しいかどうか比べる**
- **len(** 文字列 **)**…文字列の長さ
- 検索文字列 **in** 検索対象文字列 …入っているかどうか
- **startswith(** 比べる文字列 **)**…その文字列で始まるか
- **endswith(** 比べる文字列 **)**…その文字列で終わるか
- 文字列 不等号演算子 比べる文字列 …辞書順に比べる
  - < 辞書順（Unicode順）で先になっていればTrue
  - > 辞書順（Unicode順）で後になっていればTrue
  - >=, <=, !=なども用いることができる

文字の位置

- 次に出てくるリストと同じで、０文字目から始まる。
- 文字列の順番は、「0〜文字列の長さ-1」の範囲で振られている。



文字列のインデックス式

- 文字列の長さを求める
  - len(s)　# sに代入されている文字列の長さを返す
- 文字列の中から 1 文字を取り出す
  - [インデックス] を使う
  - インデックスは整数式で、範囲は0〜文字列の長さ-1
  - インデックスにマイナスを使うと最後から数える
  - マイナスの場合の範囲は、-1〜-文字列の長さ
- 例：
  - s[0]　　s[17]　　s[7]　　s[len(s) - 2]
  - s[-1]　　s[-6]　　s[- len(s)]

文字列のスライス式

- 文字列の中から範囲を指定して一定の範囲の文字列を取り出す
- スライスの記法は、以下の通り、インデックスを用いる
  - 文字列[始め:終わった次]
  - 文字列[始め:終わった次:間隔]
- 例：
  - s[0: 5]　# 最初の５文字
  - s[-5: -1]　# 最後の５文字目から４文字分
  - s[7: 12: 2]　# 1 文字飛ばしつつ
- スライスの場合省略が可能
  - s[: 5] # 0からと仮定される
  - s[-5 :] # 最後まで
  - s[:] # 初めから最後まで

文字列の検索

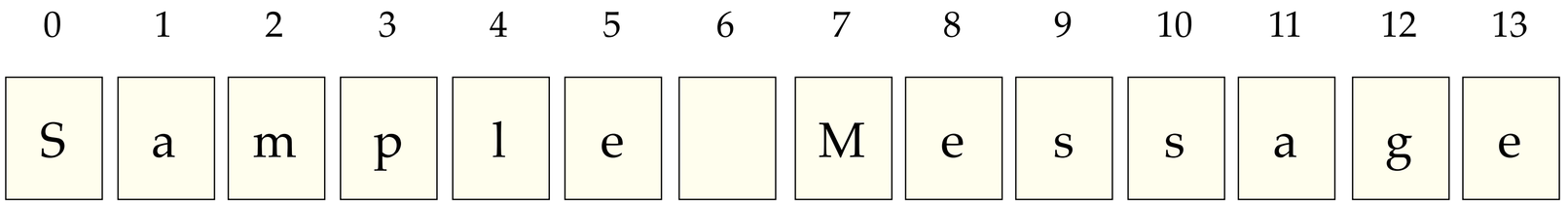
- **find**(検索文字列)…文字列の最初から検索、
  返される位置は0から、見つからなければ-1
- **find**(検索文字列, 開始位置)…開始位置から検索
- **find**(検索文字列, 開始位置, 終了位置)…終了位置の手前まで検索
- **index**(位置 [, 開始位置 [, 終了位置]]) …findと同じだが、見つからなければValueError例外扱いになる
- **rfind**(位置 [, 開始位置 [,終了位置]])…findと同じだが、文字列の最後から検索
- **rindex**(位置 [, 開始位置 [, 終了位置]])…indexと同じだが、文字列の最後から検索
- **count**(検索文字列 [, 開始位置 [,終了位置]])…検索文字列が出現する回数を返す

文字列の置換

- 文字列の足し算とスライスを使う
  - s = "This is a sample."
  - result = s[0 : 8] + "the" + s[9 :]
- **文字列は、変更不可能（immutable）なので、新しい文字列を生成するしかない。**
- replace(元の文字列, 置換文字列)…該当する文字列を置換する
  なお、３番目のパラメータで置換の最大回数を指定できる
- 置換された新しい文字列が返される
  - s = "This is a sample message for you."
  - result = s.replace("a", "the")
  - result2 = s.replace("e", "é", 2) # 2回までに制限

文字列の操作

- s = "Sample Message"
- *index* = *s*.index("Message")　# 7
- s2 = s[4 : 9]　# "le Me"
- s3 = s[- 3 :]　# "age"
- s4 = s[: 3]　　# "Sam"
- s5 = s.replace("Mess", "her ") # "Sample her age"



その他のstringの関数

- strip( ), lstrip( ), rstrip( )…不要な空白を取り除く
- ljust(文字数), rjust(文字数)…文字数分だけの文字列を確保、左右に寄せる、埋める文字は空白が標準だが、２番目のパラメータを与えることによって指定できる
- casefold( )…大文字小文字、特殊文字の区別をしないで等価性を判定する文字列を返す
- upper( ), lower( )…大文字にする、小文字にする
- capitalize( ), title( )…最初の１文字を大文字にして、後の文字を小文字にする、titleは、単語ごとにそれを行なう
- partition(区切り文字列)…文字列を区切り文字列の最初の出現位置で区切り、(区切りの前の部分, 区切り文字列, 区切りの後ろの部分)の３要素から構成されるタプルを返す
- zfill(文字数)…足りなければ、文字数を充たすだけ0を入れる

再帰で文字列を

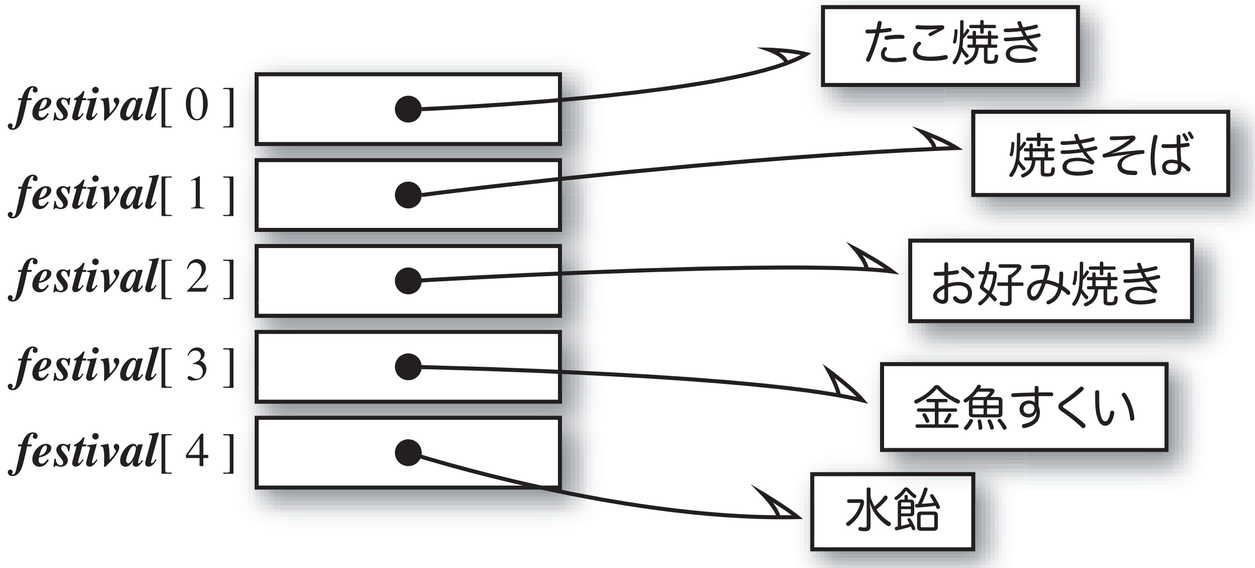
- 回文を作る例
  - 最初は、任意の文字を選ぶ
  - 前後に同じ文字（任意の文字）を追加していく
  - これを再帰で行なう

文字列のリスト

- リストの各要素が文字列になる。
- 空の文字列のリストとして初期化する
  - リスト変数名 =[""] * サイズ
    - 例：namelist =[""] * 10
- 各要素に対して、文字列を代入できる。
    - 例：namelist[0] = "Hello, Python"
    - namelist[3] = namelist[0] + " and Script"

文字列のリストの初期化

- festival = ["たこ焼き", "焼きそば", "お好み焼き", "金魚すくい", "水飴"]



文字列リストの場合

- 文字列のリスト名[インデックス].メソッド名(実パラメータ)
- 文字列のリスト名[インデックス]スライス

- festival[2].find("お好み")
- festival[2][2: 5]

- リストのスライスと文字列のスライスの組合せは機能しない
- × festival[2: 4][2: 5] # ２次元リストとされ、空のリストが返される

文字列とリストの変換

- list(文字列)… 1 文字ずつ分解したリストができる
- str(リスト)…print関数で表示されるような文字列ができる

- 文字列.split(*sep*=区切り文字)…文字列を区切り文字で区切って、リストにする
- 文字列.splitlines( )…文字列を改行文字で区切って、リストにする
- 区切り文字.join(リスト)…リストの各要素を区切り文字で繋げて文字列にする

文字列の分割

- stringクラス用のsplitメソッド
- １つの文字列を、区切りとなる文字列で分割し、文字列のリストに直すことができる
- 文字列.split(区切りとなる文字列)
  - 例：

```
tokens = "A sample message".split( " " )
# tokens = [ "A", "sample", "message" ]
for token in tokens:
    print( token )
```

文字列の統合

- 文字列のリストから、間にいれる文字列を指定して、１つの文字列に統合することができる
- "間の文字列".join(文字列のリスト)
- 例：

```
colorlist = [ "red", "blue", "green", "white" ]
    total = " and ".join( colorlist )
    # total = "red and blue and green and white"
    print( total )
```

正規文字列

- UNIXの伝統の正規文字列
- .　…　任意の一文字
- ?　…　任意の0文字か 1 文字
- *　… 0回以上の繰返し
- + … 1回以上の繰返し
- \a　… その文字自体を表わす　\.　\?　\\　*
- [abc]　…　文字のグループ化　[a-z,A-Z,0-9]　[b-x]
- [^abc]　… ^はそれは含めない
- ^ …行の先頭
- $ …行の最後

正規文字列を使った検索

- 正規文字列を扱うモジュール re
  - **import** re # 利用する前に
- 正規文字列で検索
  - マッチオブジェクト = re.search(正規文字列, 検索対象)
  - 例：matset = re.search("[0,4-9]+th", "the 45th event")
- マッチオブジェクトを用いて検索結果
  - 見つからなければNoneになっている
  - start(), end()で、見つかった先頭の位置、終了の位置を求められる
- re.findall(正規文字列, 検索対象) で検索された文字列のリストを得ることができる
  - 例: numlist = re.findall("[0-9]+", "the 123 students get 623 texts.")

正規文字列ですべての該当文字列を検索する

- 自分で繰返しとスライスを使って、検索する必要がある
- プログラミング例：

```
import re
text = "every Thursday in 2023/March/9~31: PM2:45~3:15"
pattern = "[0-9]+"

sp = 0 # start point
while True:
    result = re.search( pattern,
            text[ sp: ] )
    if result == None: break
    st, en = result.start(), result.end()
    print( text[st+sp:en+sp] )
    startpoint = en+sp
```

リスト（復習）

- 複数の値をまとめておける
  - リスト、タプルがPythonには用意されている
- リストの記述は、[ ]を用いる
  - 例：numlist = [2, 3, 5, 7, 11, 13, 17, 19, 23, 31] # 素数リスト
- リストの個々の値は、要素と呼ばれる
- Pythonでは、要素の型は異なっていても構わない
  - 例：heterogeneous = [345, True, "WOW", 45.6e-12]
- リストの要素の個数を調べるのにlen組込み関数が使える
  - 例：len(numlist)

インデックス式

- インデックスの式を用いて、要素を 1 つ取り出すことができる
- 文字列のインデックスと同じで、範囲は、
  正の整数の場合：0〜要素の個数-1
  負の整数の場合：-1 〜 - 要素の個数
- 例
  - numlist[0] # 最初の要素
  - numlist[-1] # 最後の要素
  - numlist[len(numlist) -1] # 最後の要素

スライス式

- スライスで複数の要素を取り出すことができる
- 結果は、リストとして返される
- 例：
  - numlist[2:3] # 1個の要素の場合でもリストとして返す
  - numlist[5:9] # ５番目から９番目の前まで
  - numlist[: 5] # 最初の５個の要素
  - numlist[-5:] # 最後の５個の要素
  - numlist[2:7:2] # ２番目から６番目まで、１つ飛ばし
  - numlist[::-1] # 逆順に取り出す
  - numlist[::-2] # １つ飛ばしで逆順
  - numlist[0:-1:-2] は動かない
  - numlist[1:-1][::-2]

スライス式を使ったリストへの代入

- リストの場合、スライスを使って代入ができる（文字列は変更不可能なのでエラーになる）
- 代入後は、元の部分が、代入したリストに置き換えられる。
- 例：

  a[4: 6] = [3, 4, 5, 6, 7]
  a[5: 8] = [12]

リストへの変換

- list()関数を使う
  - 書式：list(値)
  - 例：list((56, 32, 193, 23))
  - 評価は、[56, 32, 193, 23]となる
- 他のオブジェクトから、リストの形にするときにもlist関数が用いられる
  - 例：list(range(5))
  - 評価：[0, 1, 2, 3, 4]

タプル

- タプルは変更不可能なデータの連なりで、一般的に異種のデータの集まりを示す
- 丸括弧の対を使い、空のタプルを表す: ()
- カンマを使い、単要素のタプルを表す: a, または (a,)
- 項目をカンマで区切る: a, b, c または (a, b, c)
- 組み込みの tuple()関数でタプルに変換することが可能

- タプルにもインデックス・スライス記法は使える
- 変更不可能なので、タプルの個々の要素には代入はできない

リスト・タプル・文字列の共通演算

- xを式とし、sおよびtをリスト（あるいはタプル・文字列）とする
- x **in** s　あるいは　x **not in** s
  - xがsに含まれている（いないかどうか）
  - True/Falseが返される
- s + t
  - 結合された新しいリスト・タプル・文字列が返される
- s * 整数　あるいは 整数 * s
  - sが整数回繰り返されたリスト・タプル・文字列が返される
  - ただし、個々の要素がリスト・タプルなどのようなオブジェクトの場合は、共有されている可能性もあるので注意のこと

リスト・タプル・文字列の演算用の関数

- len(s)
  - 個数や長さが整数で返される
- min(s) あるいは max(s)
  - s中の要素の最小値／最大値が返される
- s.index(x)
  - s中で指定された値が何番目にあるかが返される（ただし、0番始まり）
  - s.index(x, 検索開始のインデックス)
  - s.index(x, 検索開始のインデックス, 検索終了+1のインデックス)
- s.count(x)
  - sの中に何個指定された値があるのか整数で返される

Set

- 集合型は、{ }で囲むか、set関数を使ってリストなどから生成する、**for**文を使っても生成することができる
  - 例：fruits = { "りんご", "みかん", "なし", "いちご" }
  - 例：xset = { x**2 **for** x **in** range(1, 11) **if** x % 2 == 1 }
  - 例：drinks = set(["紅茶", "コーヒー" , "ジュース"])
- 集合なので、重複項目は、除去される
- **in / not in** 演算子を使って、集合に入っているかどうかを判定
  - 例："なし" **in** fruits
- 組込み関数lenで、集合内の要素の個数を調べることができる

Setと要素

- 集合.add(要素) で、要素を追加する
- 集合.remove(要素)で、要素を削除する、ただし要素がないとエラー
- 集合.discard(要素)で、含まれていればその要素を削除する
- 集合.pop( )で、任意の要素を返し、その要素を削除する、ただし 1 つも要素がないとエラー
- 集合.clear( )で、集合の要素を全部削除する

Setでの演算

- 集合同士の演算→結果の集合が返される
  - 和集合 union　A | B
  - 差集合 difference　A - B
  - 交差集合 intersection A & B
  - 排他的論理和集合 symmetric_difference　A ^ B　≡　(A | B) - (A & B)
- 部分集合の判定→論理値が返される
  - 部分集合か is subset　A <= B
  - 真部分集合か　A < B
  - 上位集合か is superset　A >= B
  - 真上位集合か A > B

辞書（Dictionary）

- 辞書では、キーでエントリの値を引いてくることができる、ただしキーは一意である必要がある
- キー値：対応する値のペア（エントリ: entryと呼ばれる）を、{ }の中に入れることで、辞書を作ることができる。for文を利用して作成することも可能
- 例：flowers = { "rose" : "薔薇", "lily": "百合", "hydrangea": "紫陽花" }
- 例： numbers = { n+1: name **for** n, name **in** enumerate(["one", "two", "three"]) }
- dict関数を使っても、タプルを要素としてもつリストなどから辞書を作成することができる
- 例：numbers = dict(zip(["one", "two", "three"], range(1, 4)))

辞書とエントリ

- 辞書[key]…keyの値を持つvalueを返す
- 辞書[key] = value…keyとvalueの組み合わせを登録
- **del** 辞書[key]…指定されたキーのエントリを削除する
- 辞書.clear( )…全キー（エントリ）をクリアする
- key **in** 辞書 / key **not in** 辞書…keyが辞書にあるか／ないかどうか論理値で返す

辞書の一覧

- len( )…エントリの個数を返す
- keys( )…辞書にあるすべてのキーを返す
- values( )…辞書にあるすべての値を返す
- items( )…辞書にあるすべてのエントリを返す

Iterator

- 何か要素が入っている構造物（コンテナ：containerと呼ばれる）があるときに、繰返しで要素を取り出せるようにするイテレータ型が用意されている
- iter(構造物)関数を用いる
- 例：drinks = { "紅茶", "コーヒー" , "ジュース" }

  for n **in** iter(drinks) : print(n) # 要素を表示
- 例：numbers = dict(zip(["壱", "弐", "参"], range(1, 4)))

  for n **in** iter(numbers) : print(n) #キーを表示
- 辞書の場合には、iterの対象は、キーになる

tuple, dict, setもfor文で作成できる

- listの生成
  - 例：list(n**0.5 **for** n **in** [1, 4, 9, 16, 25]) → [1.0, 2.0, 3.0, 4.0, 5.0]
- tupleの生成
  - 例：tuple(n **for** n **in** range(5)) → (0, 1, 2, 3, 4)
- dictの生成
  - 例：{ n: n**2 **for** n **in** range(3, 8) } →
    {3: 9, 4: 16, 5: 25, 6: 36, 7: 49}
- setの生成
  - 例：{ n**2 **for** n **in** range(3, 8) } →
    {36, 9, 16, 49, 25}

関数への引数へのジェネレータ型

- リストやタプルを引数を取る関数に対してもfor文を使ったジェネレータ型で実引数を与えることができる
  - type(n **for** n **in** range(1, 10)) …<class 'generator'>が返される
  - sum(n **for** n **in** range(1, 10))…45が返される
  - set(n **for** n **in** range(2, 22, 2))…{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}が作られる
  - dict((str(n),n) **for** n **in** range(1, 6))…{'1': 1, '2': 2, '3': 3, '4': 4, '5': 5}が作られる

課題

- 日本語の１文を品詞に分解し、名詞・形容詞・動詞・形容動詞・助動詞・助詞（格助詞・副助詞・終助詞・接続助詞）・副詞・接続詞・代名詞・連体詞などに分類せよ。
- 、。．，「」 などの記号は入らないものとする。
- 登録されていない単語は、推定できるものは推定し、そうでないものは「未知」として出力する。
- 名詞はすべて漢字であることを前提にする。