Practicum in human intelligent information processing

Vishal Gaurav

Python

- インタプリタ
- 対話入力
- オブジェクト指向

Table of content

Pythonの実行

Python を起動する３つの方法:

対話型インタプリタ: コマンドラインで **python** と入力.

```
$python                 # Unix/Linux

or

python%                 # Unix/Linux

or

C:>python               # Windows/DOS
```

実行可能スクリプト: コマンドラインからPython スクリプトファイル(.py) を実行.

```
$python  script.py          # Unix/Linux

or

python% script.py           # Unix/Linux

or

C:>python script.py         # Windows/DOS
```

統合開発環境(IDE): PythonをGUI から実行.

Pythonスクリプトの実行

テキストエディタで以下のコードを入力し，**hello.py**として保存．通常Pythonファイルの拡張子は**.py** となる．

```
#!/usr/bin/env python
print('hello world')
```

コマンドラインを起動し **python hello.py** と入力するとプログラムが実行できる．

変数

- 変数は特定のフォーマットをもつメモリ内領域のシンボル名．そのフォーマットは領域の型である．
- 与えられた値を変えることができるので，変数と呼ばれる．
- Python の変数はその柔軟な性質により，プログラム中の任意のタイミングで任意の型に再バインドできる．

有効な変数名:

myVariable
Var1
X
x
_x

無効な変数名:

1var – wrong, begins with digit
My#var- wrong

変数名には a-z, A-Z, _, 0-9 のみが使える. Other special characters are not permitted.

変数

標準データ型.

- 数値
- 文字列
- リスト
- タプル
- ディクショナリ

```
#!/usr/bin/python
counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
print counter
print miles
print name
```

複数同時の代入

```
a=b=c=1
a, b, c = 1, 2, "john"
```

文字列とテキスト

以下のプログラムを実行してみよう:

```
#!/usr/bin/python
str = 'Hello World!'
print str # Prints complete string
print str[0] # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:] # Prints string starting from 3rd character
print str * 2 # Prints string two times
print str + "TEST" # Prints concatenated string
```

リスト

リストはPythonの複合データ型の中で最も汎用性が高い．リストは要素をカンマ(,)で区切り，括弧 [] で囲んで記述する．異なる型の要素を一つのリストにまとめることも可能である．
リスト内の要素にはスライス ([ ] と[:]) を用いてアクセスできる． インデックスは0から始まり（要素数 - 1）で終わる
可算演算子（+）はリスト同士の結合，乗算演算子（*）はリストの繰り替えしを行う演算子である．

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list # Prints complete list
print list[0] # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:] # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists
```

タプル

タプルはリストに似たもう一つのシーケンス型. カンマで区切られた複数の値から成り立つ.
リストと異なり丸括弧 ( ) で囲まれる.

リストとタプルの違い

リストは角括弧([ ])で囲まれ, サイズを変更することができるが, タプルは丸括弧 (( ))で囲まれ,
上書きができない. タプルは**read-only** なリストと考えることができる.

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print tuple # Prints complete tuple
print tuple[0] # Prints first element of the tuple
print tuple[1:3] # Prints elements starting from 2nd till 3rd
print tuple[2:] # Prints elements starting from 3rd element
print tinytuple * 2 # Prints tuple two times
print tuple + tinytuple # Prints concatenated tuple
```

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000 # Invalid syntax with tuple
list[2] = 1000 # Valid syntax with list
```

ディクショナリ

Pythonのディクショナリはハッシュテーブル型の一種である．これらはPerlの連想配列やハッシュのように動作し，キーと値のペアで成り立つ．ディクショナリのキーはどの方でもよいが,通常は数値型や文字列型を用いる．一方で，値型には任意のPythonオブジェクトを使用できる．
ディクショナリは中括弧({ })で囲まれ，角括弧 ([]) を用いて値の割り当て，読み込みを行うことができる．

```
#!/usr/bin/python
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print dict['one'] # Prints value for 'one' key
print dict[2] # Prints value for 2 key
print tinydict # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

Basic Operators

Arithmetic Operators

Operator	Description	Example
+ Addition	片方のオペランドに値を追加する.	a + b = 30
- Subtraction	左側のオペランドから右側のオペランドを引く.	a – b = -10
* Multiplication	演算子の反対側にあるオペランドをかける.	a * b = 200
/ Division	左側のオペランドを右側のオペランドで割る.	b / a = 2
% Modulus	左側のオペランドを右側のオペランドで割った余りを返す.	b % a = 0
** Exponent (べき乗)	べき乗を求める演算子.	a**b =10 to the power 20
// Floor Division	切り捨て除算を行う. 結果は一番小さい整数に丸められる.	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

a=10
b=20

もしオペランドの一つが負であれば結果は，底値となる.
すなわち小数点以下を切り上げる

Practice Exercise 1

式1-5をpythonで計算せよ

1. 7845+2345
2. 89.23-45.2
3. 56x72
4. 9846/47
5. 3^{32}

比較演算子

Operator	Description	Example
==	2つのオペランドが等しいとき条件は真となる.	(a == b) is not true.
!= or <>	2つのオペランドが等しくないとき条件は真となる.	(a<>b) is true
>	左側のオペランドが右側より大きいとき,条件は真となる.	(a > b) is not true.
<	左側のオペランドが右側より小さいとき, 条件は真となる.	(a < b) is true.
>=	左側のオペランドが右側より大きい, もしくは等しいときに条件は真となる.	(a >= b) is not true.
<=	左側のオペランドが右側より小さい, もしくは等しいときに条件は真となる.	(a <= b) is true.

a=10
b=20

代入演算子

Operator	Example and description
=	c = a + b は a + b をcに割り当てる
+=	c += a は c = c + a と同等
-=	c -= は c = c – a と同等
*=	c *= a は c = c * a と同等
/=	c /= a は c = c / a と同等
%=	c %= a は c = c % a と同等
**=	c **= a は c = c ** a と同等
//=	c //= a は c = c // a と同等

Membership Operators

Operator	Description	Example
in	ある値が，指定されたシーケンスの中にあればTrueを返し，ない場合はFalseを返す.	x in y, here in results in a 1 if x is a member of sequence y.
not in	ある値が，指定されたシーケンスの中になければTrueを返し，その他の場合はFalseを返す.	x not in y, here not in results in a 1 if x is not a member of sequence y.

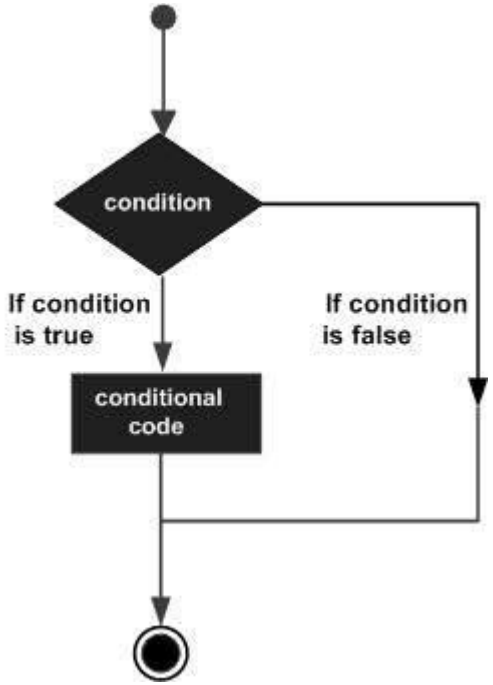
Identity Operators

Operator	Description	Example
is	２つのオブジェクトが同一オブジェクトのときTrueを返す.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	２つのオブジェクトが異なるオブジェクトが異なるときTrueを返す.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Decision making

if文は式を評価し，その結果が真のときifステートメント内の処理を実行する.

if statements
if….else statements
nested statements

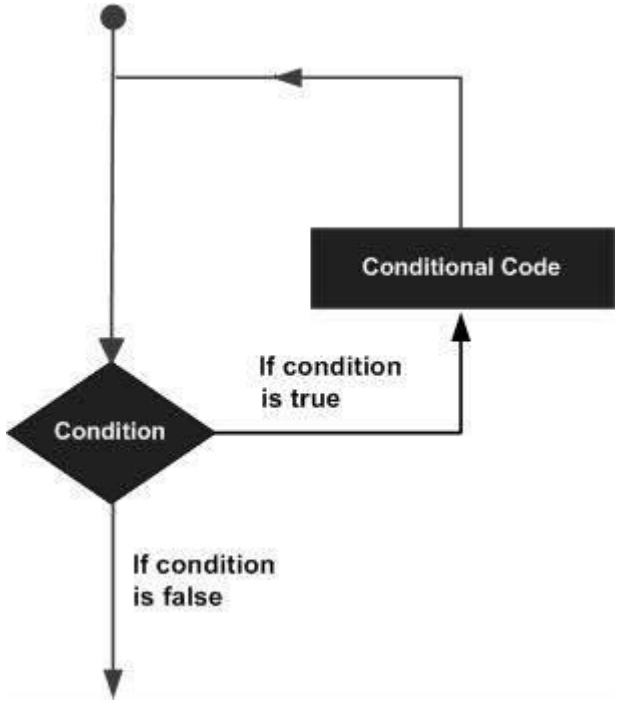



```
#!/usr/bin/python
var = 100
if ( var == 100 ) :
   print "Value of expression is 100"
print "Good bye!"
```

ループ

ループステートメントによって単数もしくは複数の処理を繰り返し実行することができる.

Loop Types
- While loop
- For loop
- Nested loops



Control statement
- Break
- continue
- pass

Forループ

処理群を指定した回数実行する.

```
for i in range(start, stop, step):
    Statements
```

```
for i in range(5):
     print i
for j in range(20,30):
    print j
for k in range(1,10,2):
    print k
```

```
#!/usr/bin/env python
a = 1
b = 1
for c in range(1,10):
    print (a)
    n = a + b
    a = b
    b = n
print ("")
```

while ループ

syntax

```
while condition:
    statements;
```

Example

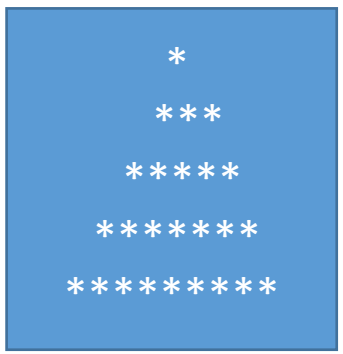
```
#!/usr/bin/python
a = 0
while a < 5:
    a += 1 # Same as a = a + 1
    print (a)
```

To add number from user input having a check condition

```
#!/usr/bin/python
a = 1
s = 0
print ('Enter Numbers to add to the sum.')
print ('Enter 0 to quit.')
while a != 0:
    print ('Current Sum: ', s)
    a = raw_input('Number? ')
    a = float(a)
    s += a
print ('Total Sum = ',s)
```

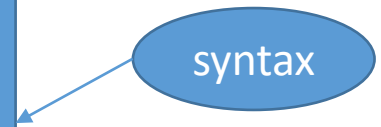
Practice Exercise 2

以下の図を描画するプログラムを作成せよ.



Function　_関数

```
def function_name( parameters ):
        "function_docstring"
        function_suite
        return [expression]
```



関数は一つの関連した動作を行う，体系的で再利用可能なコードのまとまりである．
アプリケーションに，より良いモジュール性と高度なコードの再利用性を提供する

必要な機能を提供するために関数を定義することができる． Pythonで関数を定義するシンプルな規則は以下のとおりである．

- 関数は**def**キーワードから始まり，関数名，引数 (( )) とつづく．
- 入力引数は丸括弧内(( ))に配置される．丸括弧内でパラメータを定義することもできる．
- 最初のステートメントは任意である – 関数のドキュメンテーション文字列 (docstring)
- 関数内のコードブロックはコロン(:) から始まり，インデントされる．
- return [expression] ステートメントは関数を終了し，必要に応じて呼び出し元に式を返す． 引数なしのreturnステートメントはreturn None と同じ．(Noneオブジェクトを返す)

Example

```
def printme( str ):
   "This prints a passed string into this function"
   print str
   return
```

関数の呼び出し

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

参照渡し

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print "Values inside the function: ", mylist
   return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Function Arguments:

Required Arguments

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;
# Now you can call printme function
printme(str="My String")
```

Keyword arguments

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;
# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

Order is not important

Default arguments

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;
# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Variable-length arguments

```
#!/usr/bin/python
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: " print arg1
   for var in vartuple:
      print var
   return;
# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

匿名関数

これらの関数は標準的な方法で宣言されない.
小さな匿名関数を作成するために*lambda* キーワードを使用する.

Syntax

```
lambda [arg1 [,arg2,.....argn]]:expression
```

Example

```
#!/usr/bin/python
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;
# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

- ラムダ式はいくつでも引数をとることができるが，返り値一つのみである.
  コマンドや複数の式を含めることはできない.

Practice Exercise 3

1. フィボナッチ数列を計算する関数を書きなさい
Write a function to calculate Fibonacci series?

String revisited

Strings は Python で最もポピュラーな型．文字を引用符で囲んで作成する．
Pythonはシングルクォート(‘ ’)とタブルクォート(“ ”)を同様に扱う．
e.g.
var1=“Hello World!”
var2=‘Python Programming’
Accessing values in string
Pythonは文字型(Char)をサポートしていない; これらは長さ1の文字列として扱われる．

```
#!/usr/bin/python
var1 = 'Hello World!'
var2 = "Python Programming"
print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Updating String:
変数に別の文字列を再割り当てすることによって，既存の文字列を更新することができる．
We cannot update an string location wise. – Pythonの文字列は後から変更できない．

Strings revisited

Common String Methods in Python

Methods	Description
stringVar.count('x')	文字列内の’x’の数を数える
stringVar.find('x')	文字‘x’の位置を返す
stringVar.lower()	stringVarのコピーを小文字にして返す (this is temporary)
stringVar.upper()	stringVarのコピーを大文字にして返す (this is temporary)
stringVar.replace('a', 'b')	文字列内の’a’をすべて’b’に置き換える
stringVar.strip()	文字列の先頭と末尾から空白を取り除く

3重クオート:
複数行にわたるコメントを利用可能.
It is basically used when we want to allow strings to span multiple lines, including verbatim NEWLINEs, TABs, and other special characters.
3つの連続したシングルクオートもしくはダブルクオートによって成り立つ.
e.g.

```
#!/usr/bin/python

para_str = """this is a long string that is made up of several lines and non-printable characters such as
TAB ( ¥t ) and they will show up that way when displayed. NEWLINEs within the string, whether
explicitly given like this within the brackets [ ¥n ], or just a NEWLINE within the variable assignment
will also show up. """

print para_str
```

Lists 再考

リストはPythonで最も汎用性が高い型で，コンマ区切りの値（要素）の並びを角括弧で囲んだものとして書き表される．リストは同じ型の集合である必要はない．

単純にコンマ区切りの値を角括弧で囲むことでリストを作成できる．

Example

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Accessing Values in Lists
Use Square Bracket ([]) for slicing along with the index.
Example:

```
#!/usr/bin/python
List_1=[‘physics’, ‘chemistry’, 1997, 2000]
List_2= [1,2,3,4,5,6,7]
print “list1[0]:”, List_1[0]
print “list2[1:5]”,List_2[1:5]
```

Updating List
We can update single or multiple entries in a list by giving slice on the left hand of the assignment operator
Example:

```
#!/usr/bin/python
list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2 : "
print list[2]
list[2] = 2001;
print "New value available at index 2 : " print list[2]
```

List 要素の削除:
要素の位置が既知であればdel ステートメントを用いてリストの要素を削除できる，もしくは removeステートメントを用いてリストからある要素を削除できる.

Example:

```
#!/usr/bin/python
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.remove('xyz');
print "List : ", aList
aList.remove('abc');
print "List : ", aList
```

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
```

基本的なリスト演算子

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Indexing and スライシング

リストはシーケンスであるためスライシングとindexingは文字列と同じように作用する.

Assume:

L=[‘spam’, ‘Spam’, ‘SPAM!’]

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

組み込み関数とメソッド

SN	Function with Description
1	cmp(list1, list2) リスト内の要素同士を比較する.
2	len(list) リストの要素数を返す.
3	max(list) リスト内で最大の要素を返す.
4	min(list) リスト内で最小の要素を返す.
5	list(seq) タプルをリストへ変換する.

SN	Methods with Description
1	list.append(obj) オブジェクト obj をリストに追加する.
2	list.count(obj) リスト内に含まれる obj の数を返す.
3	list.extend(seq) seqの要素を listと連結したリストを返す.
4	list.index(obj) リスト内の要素objのインデックスのうち最小のものを返す.
5	list.insert(index, obj) リスト内の指定した位置(index)に要素を挿入する.
6	list.pop(obj=list[-1]) リスト内の指定した要素を削除して, その要素を返す.
7	list.remove(obj) リストから値がobj となる(最初の)要素を削除する.
8	list.reverse() リストの順番を逆順にする.
9	list.sort([func]) リスト内の要素をソートする. 比較関数が与えられた場合はそれを使用する.

Practice Exercise 4

1. Find the max and min value in the following list.
   次のリストの最小値と最大値を求めよ.

List_1=[23,1,34,79,45,56,88,5,90]

2. Compare two user given list.
   任意のリスト2つを比較せよ.
3. Reverse a user given list.
   任意のリストを逆順にせよ. (例: [1,2,3] → [3,2,1])
4. Sort a user given list.
   任意のリストを並び替えよ.
5. Convert a tuple in a list.
   タプルをリストに変換せよ

Dictionary 再考

ディクショナリの値読み取り:
[] でキーを指定し値を取得する.
例)

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

ディクショナリ更新：
新規エントリ・key-valueペアの追加，既存のエントリの修正，または削除によって以下のようにDictionary を更新できる.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Dictionary要素の削除:
特定のディクショナリ要素やディクショナリ自体を削除することができる.
E.g.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Built in Dictionary function:

SN	Function with Description
1	cmp(dict1, dict2) ２つのディクショナリを比較する
2	len(dict) ディクショナリの長さを返す．これはディクショナリ内の要素数に等しい．
3	str(dict) ディクショナリを表示可能な文字列型へ変換する．
4	type(variable) 与えられたvariableの型を返す．ディクショナリが渡された場合，ディクショナリ型が返される．

Dictionary Methods

SN	Methods with Description
1	dict.clear() dict内のすべての要素を削除.
2	dict.copy() 辞書の浅いコピーを返す.
3	dict.fromkeys(seq[, value]) seqをキーとしvalueを値に設定した新しい辞書を作成する.
4	dict.get(key, default=None) keyが辞書にあればそれに対する値を返す. ない場合defaultを返す.
5	dict.has_key(key) keyが辞書にあればtrueを返す. それ以外の場合false
6	dict.items() 辞書のコピーをタプル(key, value)のリストとして返す.
7	dict.keys() 辞書のキーのリスト(のコピー)を返す.
8	dict.setdefault(key, default=None) get()と類似しているがキーが存在しない場合, defaultが挿入される
9	dict.update(dict2) dictに辞書dict2のkey-valueペアを登録する.
10	dict.values() 辞書の値のリスト(のコピー)を返す.

Examples

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7};
print "Variable Type : %s" % type (dict)
```

```
#!/usr/bin/python
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Mahnaz', 'Age': 27};
dict3 = {'Name': 'Abid', 'Age': 27};
dict4 = {'Name': 'Zara', 'Age': 7};
print "Return Value : %d" % cmp (dict1, dict2)
print "Return Value : %d" % cmp (dict2, dict3)
print "Return Value : %d" % cmp (dict1, dict4)
```

```
#!/usr/bin/python
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print "New Dictionary : %s" % str(dict)
dict = dict.fromkeys(seq, 10)
print "New Dictionary : %s" % str(dict)
```

Input and Output

Python はユーザーからの入力を受け付ける2つの関数を持つ:

- input()
- raw_input()

raw_input() は文字列データの入力を要求し(ended with a newline), 単純にその文字列を返す. ユーザーがデータを入力する前に表示するプロンプトを, 引数に渡すことができる. E.g.

```
x = raw_input('What is your name?')
print ('Your name is ' + x)
```

input() はデータの文字列を読み取るのにraw_inputを使用し, それがPythonプログラムであるかのように実行, そして結果の値を返す.
基本的にこれはリストや辞書型のように整形された入力を受け取る.
例)

```
x = input('What are the first 10 perfect squares? ')
What are the first 10 perfect squares?
map(lambda x: x*x, range(10))
```

Message displayed on screen

User input for evaluation

raw_input()から返される文字列を次のようなイディオムでPythonの型へ変換する.

```
x = None
while not x:
    try:
        x = int(raw_input())
    except ValueError:
        print 'Invalid Number'
```

Assignment to be submitted

1. Create a program to count by prime numbers. Ask the user to input a number, then print each prime number up to that number.
   入力された数までの素数を数えるプログラムを作成せよ.

2. Instruct the user to pick an arbitrary number from 1 to 100 and proceed to guess it correctly within seven tries. After each guess, the user must tell whether their number is higher than, lower than, or equal to your guess.
1から100までの数から任意の数を選択しその値をコンピュータに推測させるプログラムを作成せよ．試行回数は7回とし，各試行の後には予測値が真値に対してhigh かlow か equal なのかを出力せよ.
Hint: use binary search. Make an array or list of 100 numbers.
ヒント: リストか配列を作成して，二分探索を使用する.

3. Write a program to reverse the ordering of words in a string.
   文字列中の単語の順番を入れ替えるプログラムを作成せよ.
e.g. Hello world, this is python!
   'python! is this world, Hello‘
Hint: use string function
   split() and join()

Please submit you assignment to gaurav-vishal@edu.brain.kyutech.ac.jp with subject “Assignment for PHIIP” file name should be your student_number also please write your name in the scripts.