CSc 120
Introduction to Computer Programing II

Adapted from slides by
Dr. Saumya Debray

01-a: Python review

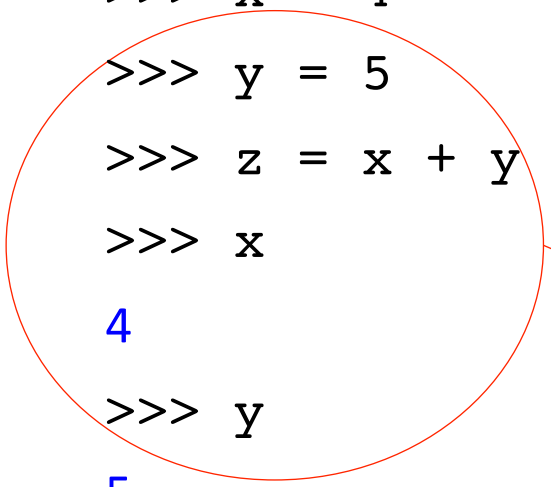
python review: variables, expressions, assignment

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

python basics

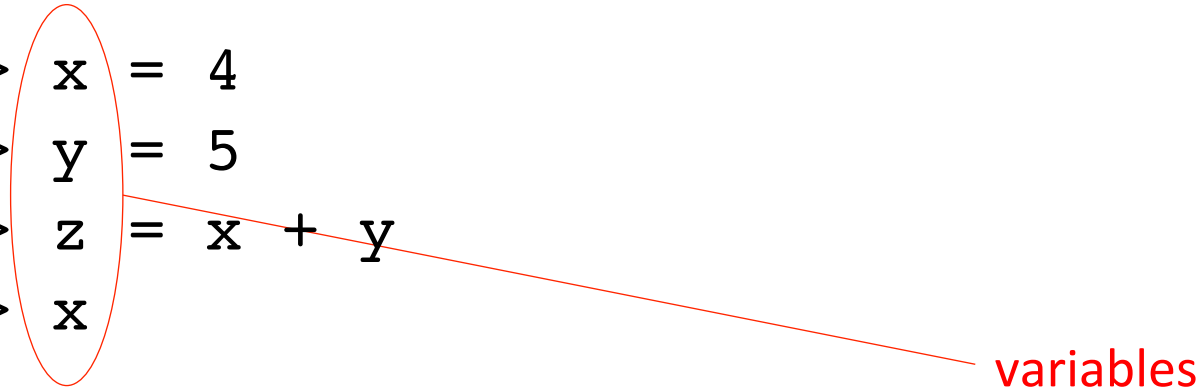
```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```



>>> : python interpreter's prompt
black: user input (keyboard)
blue: python interpreter output

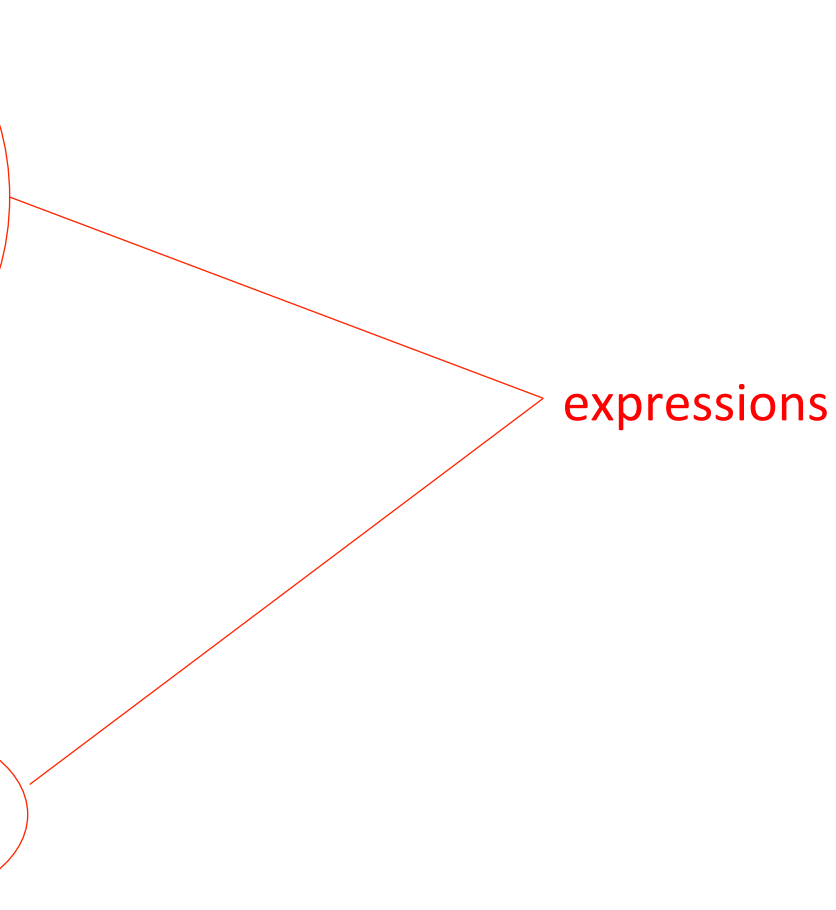
python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```



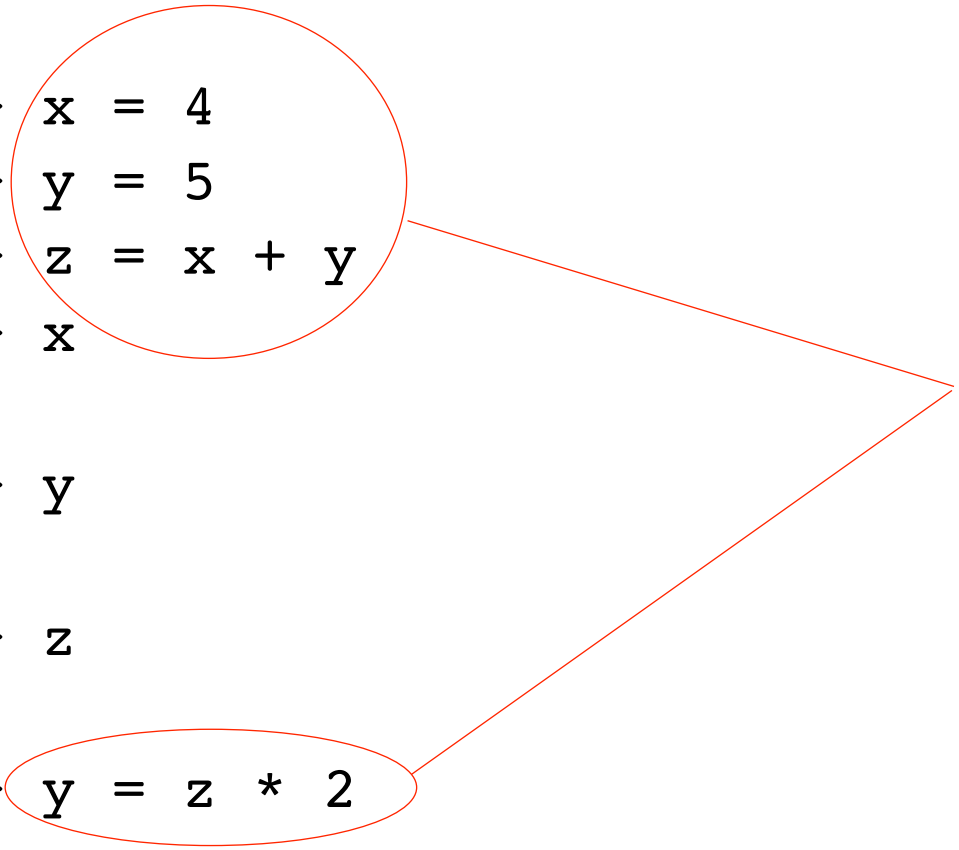
python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```



python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```



assignment statements

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

typing in an expression causes its value to be printed

python basics

```
>>> x = 4
>>> y = 5
>>> z = x + y
>>> x
4
>>> y
5
>>> z
9
>>> y = z * 2
>>> y
18
>>>
```

- variables:
  - names begin with letter or '_'
  - don't have to be declared in advance
    - type determined at runtime
- expressions:
  - all the usual arithmetic operators

Multiple (aka parallel) assignment

```
>>>
>>> x, y, z = 11, 22, 33
>>> x
11
>>> y
22
>>> z
33
>>>
```

Assigns to multiple variables at the same time

$x_1, x_2, ..., x_n = exp_1, exp_2, ..., exp_n$

Behavior:

1. $exp_1, ..., exp_n$ evaluated (L-to-R)
2. $x_1, ..., x_n$ are assigned (L-to-R)

Comparison and Booleans

```
>>> x, y, z = 11, 22, 33
>>> x
11
>>> y
22
>>> z
33
>>> x < y
True
>>> y == z
False
```

Comparision operations:

<, >, ==, >=, <=, !=

Lower precedence than arithmetic operations.

Yield boolean values:

True

False

EXERCISE

```
>>> x = 3
>>> y = 4
>>> z = (2*x – 1 == y+1)
>>> z
```

← *what value is printed out for* z?

EXERCISE

```
>>> x = 3
>>> y = 4
>>> sum, diff, prod  =  x + y,  x – y,  x * y
>>> prod+diff
```

← *what is the value printed out*?

python review: reading user input I: input()

Reading user input I: **input()**

```
>>> x = input()
13579
>>> x
'13579'
>>> y = input('Type some input: ')
Type some input: 23
>>> y
'23'
>>> z = input('More input: ')
More input: 567
>>> z
'567'
>>>
```

Reading user input I: **input()**

```
>>> x = input()
13579
>>> x
'13579'
>>> y = input('Type some input: ')
Type some input: 23
>>> y
'23'
>>> z = input('More input: ')
More input: 567
>>> z
'567'
>>>
```

input statement:

- reads input from the keyboard
- returns the value read
  - (a string)

Reading user input I: **input()**

```
>>> x = input()
13579
>>> x
'13579'
>>> y = input('Type some input: ')
Type some input: 23
>>> y
'23'
>>> z = input('More input: ')
More input: 567
>>> z
'567'
>>>
```

input statement:

- reads input from the keyboard
- returns the value read
  - (a string)

- takes an optional argument
  - if provided, serves as a prompt

Reading user input I: **input()**

```
>>>
>>> x = input()
12
>>> x
'12'
>>> y = x / 2
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    y = x / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

the value read in is represented as a string

- string ≡ sequence of characters

Reading user input I: **input()**

```
>>>
>>> x = input()
12
>>> x
'12'
>>> y = x / 2
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    y = x / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

the value read in is represented as a string

- string ≡ sequence of characters

- TypeError: indicate an error due to wrong type

Reading user input I: **input()**

```
>>>
>>> x = input()
12
>>> x
'12'
>>> y = x / 2
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    y = x / 2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>> y = int (x) / 2
>>> y
6.0
>>>
```

the value read in is represented as a string

- string ≡ sequence of characters
- TypeError: indicates an error due to a wrong type

- Fix: explicit type conversion

python review: basics of strings

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
```

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>>
```

either single-quotes (at both ends)
or double-quotes (at both ends)

Basics of strings

```
>>> text = input('Enter a string: ')
Enter a string: abcdefghi
>>> text
'abcdefghi'
>>> text[0]
'a'
>>> text[1]
'b'
>>> text[27]
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    text[27]
IndexError: string index out of range
>>>
```

a string is a sequence (array) of characters

- we can index into a string to get the characters

Basics of strings

```
>>> text = input('Enter a string: ')
Enter a string: abcdefghi
>>> text
'abcdefghi'
>>> text[0]
'a'
>>> text[1]
'b'
>>> text[27]
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    text[27]
IndexError: string index out of range
>>>
```

a string is a sequence (array) of characters

- we can index into a string to get the characters

indexing beyond the end of the string gives an **`IndexError`** error

Basics of strings

```
>>> text = input('Enter a string: ')
Enter a string: abcdefghi
>>> text
'abcdefghi'
>>> text[0]
'a'
>>> text[1]
'b'
>>> text[27]
Traceback (most recent call last):
  File "<pyshell#153>", line 1, in <module>
    text[27]
IndexError: string index out of range
>>>
```

a string is a sequence (array) of characters

- we can index into a string to get the characters
- each character is returned as a string of length 1

Intuitively, a *character* is a single letter, digit, punctuation mark, etc.

E.g.: 'a'
'5'
'$'

Basics of strings

```
>>> x = '0123456789'
>>>
>>> x[0]
'0'
>>> x[1]
'1'
>>> x[2]
'2'
>>>
>>> x[-1]
'9'
>>> x[-2]
'8'
>>> x[-3]
'7'
>>>
```

x[*i*] : if *i* ≥ 0 (i.e., non-negative values):

- indexing is done from the beginning of the string
- the first letter has index 0

x[*i*] : if *i* < 0 (i.e., negative values):

- indexing is done from the end of the string
- the last letter has index -1

Basics of strings

```
>>> x = '0123456789'
>>>
>>> x[0]
'0'
>>> x[1]
'1'
>>> x[2]
'2'
>>>
>>> x[-1]
'9'
>>> x[-2]
'8'
>>> x[-3]
'7'
>>>
```

x[*i*] : if *i* ≥ 0 (i.e., non-negative values):

- indexing is done from the beginning of the string
- the first letter has index 0

x[*i*] : if *i* < 0 (i.e., negative values):

- indexing is done from the end of the string
- the last letter has index -1

EXERCISE

```
>>> x = 'a'
>>> x == x[0]
```

what do you think will be printed here?

EXERCISE

```
>>> x = 'apple'
>>> x[2] == x[-2]
```

what do you think will be printed here?

Basics of strings

```
>>>
>>> x = "5"
>>> x
'5'
>>> x == 5
False
>>>
```

Inside a computer, a character is represented as a number (its "ASCII value")

Basics of strings

```
>>>
>>> x = "5"
>>> x
'5'
>>> x == 5
False
>>>
```

Inside a computer, a character is represented as a number
(its "ASCII value")

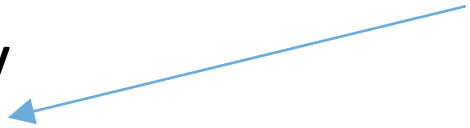
the ASCII value of a digit is not the same as the digit itself:

'5' ≠ 5

EXERCISE

```
>>> x = 27
>>> y = 'x'
>>> x == y
```

What do you think will be printed here?
Why?



Basics of strings

```
>>> x = input()
abcDE_fgHIJ_01234
>>> x
'abcDE_fgHIJ_01234'
>>>
>>> len(x)
17
>>> y = x.lower()
>>> y
'abcde_fghij_01234'
>>>
>>> x = y.upper()
>> x
'ABCDE_FGHIJ_01234'
>>>
```

len(x) : length of a string x

Basics of strings

```
>>> x = input()
abcDE_fgHIJ_01234
>>> x
'abcDE_fgHIJ_01234'
>>>
>>> len(x)
17
>>> y = x.lower()
>>> y
'abcde_fghij_01234'
>>>
>>> x = y.upper()
>> x
'ABCDE_FGHIJ_01234'
>>>
```

len(x) : length of a string x

x.lower(), x.upper() : case conversion on the letters in a string x

- note that non-letter characters are not affected

Basics of strings

```
>>> x = input()
abcDE_fgHIJ_01234
>>> x
'abcDE_fgHIJ_01234'
>>>
>>> len(x)
17
>>> y = x.lower()
>>> y
'abcde_fghij_01234'
>>>
>>> x = y.upper()
>> x
'ABCDE_FGHIJ_01234'
>>>
```

len(x) : length of a string x

x.lower(), x.upper() : case conversion on the letters in a string x

- note that non-letter characters are not affected

Python supports a wide variety of string operations

- see www.tutorialspoint.com/python3/python_strings.htm

Basics of strings

```
>>> x = input()
abcdefgh
>>> x
'abcdefgh'
>>> x[3]
'd'
>>>
>>> x[3] = 'z'
Traceback (most recent call last):
  File "<pyshell#193>", line 1, in <module>
    x[3] = 'z'
TypeError: 'str' object does not support item assignment
>>>
```

Basics of strings

```
>>> x = input()
abcdefgh
>>> x
'abcdefgh'
>>> x[3]
'd'
>>>
>>> x[3] = 'z'
Traceback (most recent call last):
  File "<pyshell#193>", line 1, in <module>
    x[3] = 'z'
TypeError: 'str' object does not support item assignment
>>>
```

strings are *immutable*, i.e., cannot be modified or updated

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = 'efgh'
>>> y == z
True
>>> x == y
False
>>>
>>> w = x + y
>>> w
'abcdefgh'
>>>
>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
```

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = 'efgh'
>>> y == z
True
>>> x == y
False
>>>
>>> w = x + y
>>> w
'abcdefgh'
>>>
>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
```

+ applied to strings does concatenation

'*' applied to strings:

- does repeated concatenation *if one argument is a number*
- generates an error otherwise

Basics of strings

```
>>> x = "abcd"
>>> y = 'efgh'
>>> z = 'efgh'
>>>
>>> w = x + y
>>> w
'abcdefgh'
>>>
>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
>>> x - y
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    x - y
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>>
```

+ applied to strings does concatenation

* applied to strings:

- does repeated concatenation *if one argument is a number*
- generates an error otherwise

not all arithmetic operators carry over to strings

Basics of strings

```
>>> x = "abcdefg"
>>> y = 'hijk'
>>>
>>> x[3:6]
'def'
>>> x[2:5]
'cde'
>>>
>>> x[:2]
'ab'
>>> x[4:]
'efg'
>>> x[4:] + y[:2]
'efghi'
```

slicing: produces substrings

- characters from 3 (included) to 6 (excluded)
- characters from 2 (included) to 5 (excluded)

- characters from the beginning to 2 (excluded)
- characters from 4 (included) to the end

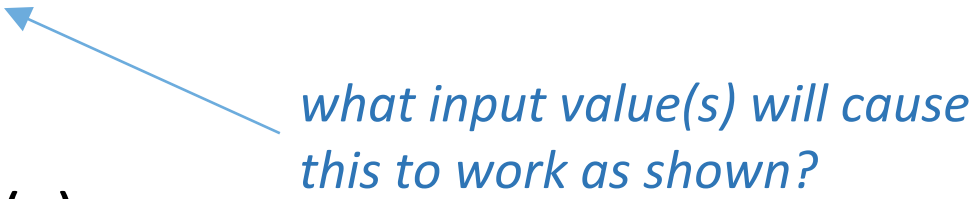
EXERCISE

```
>>> x = "whoa!"
>>> y = x[2] * len(x)
>>> z = x[3] + x[0] + y
>>> z
```

what is printed here?

EXERCISE

```
>>> x = input()
>>> y = x + x
>>> int(x) == int(y)
True
```



python review: conditionals

Conditional statements: **if/elif/else**

```
>>> var1 = input()
100
>>> var2 = input()
200
>>> x1 = int(var1)
>>> x2 = int(var2)
>>>
>>> if x1 > x2:
        print('x1 is bigger than x2')
elif x1 == x2:
        print('x1 and x2 are equal')
else:
        print('x1 is smaller than x2')
x1 is smaller than x2
>>>
```

Conditional statements: if/elif/else

```
>>> var1 = input()
100
>>> var2 = input()
200
>>> x1 = int(var1)
>>> x2 = int(var2)
>>>
>>> if x1 > x2:
        print('x1 is bigger than x2')
elif x1 == x2:
        print('x1 and x2 are equal')
else:
        print('x1 is smaller than x2')
x1 is smaller than x2
>>>
```

- **if**-statement syntax:

if *BooleanExpr* **:**
    stmt
    …
elif *BooleanExpr* **:**
    stmt
    …
elif …
    …
else:
    stmt
    …

elifs are optional (use as needed)

Conditional statements: **if/elif/else**

```
>>> var1 = input()
100
>>> var2 = input()
200
>>> x1 = int(var1)
>>> x2 = int(var2)
>>>
>>> if x1 > x2:
        print('x1 is bigger than x2')
elif x1 == x2:
        print('x1 and x2 are equal')
else:
        print('x1 is smaller than
x2')
x1 is smaller than x2
>>>
```

- **if**-statement syntax:

```
if BooleanExpr :
        stmt
        …
elif BooleanExpr :
        stmt
        …
elif …
        …
else:
        stmt
        …
```

elifs are optional (use as needed)

else is optional

python review: while loops

Loops I: **while**

```
>>> n = input('Enter a number: ')
Enter a number: 5
>>> limit = int(n)
>>> i = 0
>>> sum = 0
>>> while i <= limit:
        sum += i
        i += 1

>>> sum
15
>>>
```

Loops I: **while**

```
>>> n = input('Enter a number: ')
Enter a number: 5
>>> limit = int(n)
>>> i = 0
>>> sum = 0
>>> while i <= limit:
        sum += i
        i += 1

>>> sum
15
>>>
```

- **while**-statement syntax:

  while *BooleanExpr* **:**
  $stmt_1$
  …
  $stmt_n$

- $stmt_1$ … $stmt_n$ are executed repeatedly as long as *BooleanExpr* is True

EXERCISE

```
>>> text = "To be or not to be, that is the question."
>>> c = "o"
```

Write the code to count the number of times c occurs in text.

python review: lists (aka arrays)

Lists

```
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>> len(x)
4
>>> x[2] = 'newitem3'
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
```

Lists

```
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>> len(x)
4
>>> x[2] = 'newitem3'
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
```

a list (or array) is a sequence of values

EXERCISE

```
>>> x = [ “abc”, “def”, “ghi”, “jkl” ]
>>> x[1] + x[-1]
```

Lists

```
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>> len(x)
4
>>> x[2] = 'newitem3'
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
```

a list (or array) is a sequence of values

accessing list elements (i.e., indexing), computing length: similar to strings

- non-negative index values (≥ 0) index from the front of the list
  - the first element has index 0
- negative index values index from the end of the list
  - the last element has index -1

Lists

```
>>> x = [ 'item1', 'item2', 'item3', 'item4' ]
>>>
>>> x[0]
'item1'
>>> x[2]
'item3'
>>> len(x)
4
>>> x[2] = 'newitem3'
>>> x
['item1', 'item2', 'newitem3', 'item4']
>>> x[1:]
['item2', 'newitem3', 'item4']
>>> x[:3]
['item1', 'item2', 'newitem3']
```

a list (or array) is a sequence of values

accessing list elements (i.e., indexing), computing length: similar to strings

lists are *mutable*, i.e., can be modified or updated

- different from strings

slicing : similar to strings

Lists

```
>>> x = [11, 22, 33]
>>> y = [44, 55, 66, 77]
>>>
>>> x + y
[11, 22, 33, 44, 55, 66, 77]
>>>
>>>
>>> x * 3
[11, 22, 33, 11, 22, 33, 11, 22, 33]
>>>
```

concatenation (+) : similar to strings

multipilication (*) similar to strings

python review: functions

Functions

- **`def`** *fn_name* (arg_1 , …, arg_n)
  - defines a function *fn_name* with n arguments arg_1 , …, arg_n
- **`return`** *expr*
  - optional
  - returns the value of the expression *expr* to the caller
- *fn_name*($expr_1$, …, $expr_n$)
  - calls *fn_name* with arguments $expr_1$, …, $expr_n$

Functions

```
>>> def double(x):
        return x + x
>>> double(7)
14
>>>
>>> def num_occurences(text, c):
        n, i = 0, 0
        while i < len(text):
            if text[i] == c:
                n += 1
            i += 1
        return n
>>> num_occurences("To be or not to be, that is the question.", "o")
5
```

- **`def`** *fn_name* (arg_1 , …, arg_n)
  - defines a function *fn_name* with n arguments arg_1 , …, arg_n

- **`return`** *expr*
  - optional
  - returns the value of the expression *expr* to the caller

Lists of Lists

```
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
>>>
>>>
>>> >>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'],
['hh', 'ii', 'jj']]
>>> >>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
```

a list can consist of elements of many types, including lists

a list of lists is called a 2-d list

Lists of Lists

```
>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]
>>>
>>>
>>> >>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> >>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
```

a list can consist of elements of many types, including lists

a list of lists is called a 2-d list

if the number of rows and columns are equal, it is a grid

Lists of Lists

```
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
>>> y[0]
['aa', 'bb', 'cc']
>>> y[1]
['dd', 'ee', 'ff']
>>> y[2]
['hh', 'ii', 'jj']
>>>
>>> len(y)
3
>>> len(y[0])
3
>>>
```

a list can consist of elements of many types, including lists

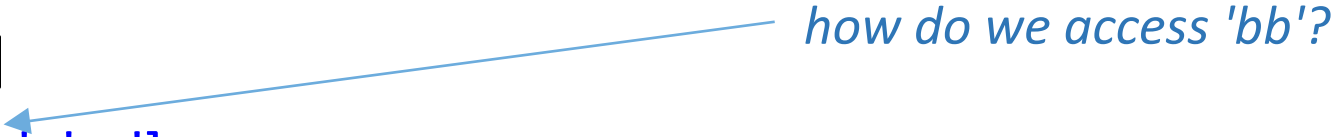
a list of lists is called a 2-d list

if the number of rows and columns are equal, it is a grid

*must check the length of each row

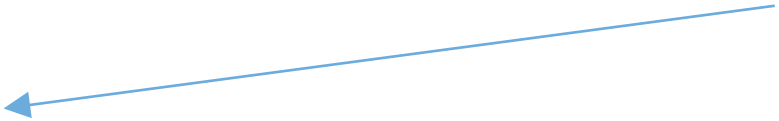
EXERCISE

```
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>>
>>> y[0]
['aa', 'bb', 'cc']
>>> y[1]
['dd', 'ee', 'ff']
>>> y[2]
['hh', 'ii', 'jj']
>>>
```



EXERCISE

```
>>> x = [ [1,2,3], [10,20,30], [100,200, 300]]
>>> x
[[1, 2, 3], [10,20,30], [100,200,300]]
>>>
>>>
```



write the code to sum the first column of x