CSc 120
Introduction to Computer Programming II

Adapted from slides by Dr. Saumya Debray

01-a: Python review
python review: variables, expressions, assignment
python basics

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

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

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

expressions
python basics

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

```python
>>> 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
>>> 

• 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

```python
>>> x, y, z = 11, 22, 33
>>> x
11
>>> y
22
>>> z
33
```
Comparison and Booleans

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

Comparison operations:
<, >, ==, >=, <=, !=

Lower precedence than arithmetic operations.

Yield boolean values:
True
False
EXERCISE

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

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

input()
Reading user input I: `input()`

```python
>>> 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'
>>>
Reading user input I: `input()`

```python
>>> 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()`

```python
>>> 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'
```
Reading user input \texttt{l: \texttt{input}()}

>>> 
>>> x = \texttt{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' 

• String \equiv sequence of characters  
• TypeError: indicate an error due to wrong type
Reading user input `input()`

```python
>>> 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
```
python review: basics of strings
Basics of strings

```python
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>> 
```
Basics of strings

```python
>>> x = "abcd"
>>> y = 'efgh'
>>> z = "efgh"
>>> 
```
either single-quotes (at both ends)
or double-quotes (at both ends)
Basics of strings

```python
>>> 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
```
Basics of strings

```python
>>> 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
```
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

```python
>>> x = '0123456789'

>>> x[0]
'0'
>>> x[1]
'1'
>>> x[2]
'2'

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

- **x[i]**: if $i \geq 0$ (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

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

- **x[i]**: if $i \geq 0$ (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

```python
>>> x = 'a'
```
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

```python
>>> 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

```python
>>> 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

```python
>>> x = input()
abcDE_fgHlJ_01234
>>> x
'abcDE_fgHlJ_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

```python
def main():
    x = input()
    abcDE_fgHIJ_01234
    >>> x
    '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
Basics of strings

```python
>>> x = input()
abcdefg
>>> x
'abcdefg'
>>> 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

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

>>> 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 = "abcd"
>>> y = 'efgh'
>>> z = 'efgh'
>>> y == z
True
>>> x == y
False

>>> w = x + y
>>> w
'abcdefgh'

>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'
Basics of strings

```python
>>> x = "abcd"
>>> y = 'efgh'
>>> z = 'efgh'
>>> y == z
True
>>> x == y
False
>>> w = x + y
>>> w
'abcdefg'  
>>> 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'

+ applied to strings does concatenation

* applied to strings:
  • does repeated concatenation if *one argument is a number*
  • generates an error otherwise

>>> w = x + y
>>> w
'abcdefgh'

>>> u = x * 5
>>> u
'abcdabcdabcdabcdabcd'

not all arithmetic operators carry over to strings

>>> x - y
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    x - y
  x - y
TypeError: unsupported operand type(s) for -: 'str' and 'str'

>>>
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'
EXERCISE

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

What is printed here? `awooooo`
EXERCISE

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

What input value(s) will cause this to work as shown?
python review: conditionals
Conditional statements: if/elif/else

```python
>>> 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

```python
>>> 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')
elif ...
else:
    print('x1 is smaller than x2')

x1 is smaller than x2

>>> 
```

• **if-statement syntax:**

```python
if BooleanExpr :
    stmt
    ...
elif BooleanExpr :
    stmt
    ...
elif ...
else:
    stmt
    ...
```

*elifs are optional (use as needed)*
Conditional statements: if/elif/else

```python
>>> 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')
```

• if-statement syntax:
```python
if BooleanExpr:
    stmt
...
elif BooleanExpr:
    stmt
...
else:
    stmt
```

elifs are optional (use as needed)
else is optional

x1 is smaller than x2
>>>
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  
>>>
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

```python
>>> 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

```python
>>> x = ["abc", "def", "ghi", "jkl"]
```

```python
>>> x[1] + x[-1]
```

What do you think will be printed here?
Lists

```python
>>> 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 ($\geq 0$) 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

```python
>>> 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]
```
python review: functions
Functions

• **def** `fn_name ( arg₁, ..., argₙ )`
  - defines a function `fn_name` with `n` arguments `arg₁, ..., argₙ`

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

• `fn_name(expr₁, ..., exprₙ)`
  - calls `fn_name` with arguments `expr₁, ..., exprₙ`
Functions

>>> def double(x):
    return x + x
>>> double(7)
14

>>> def num_occurrences(text, c):
    n, i = 0, 0
    while i < len(text):
        if text[i] == c:
            n += 1
        i += 1
    return n

>>> num_occurrences("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

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

>>> x = [ [1,2,3], [4], [5, 6]]
>>> x
[[1, 2, 3], [4], [5, 6]]

a list of lists is called a 2-d list

>>> y = [ ['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
>>> y
[['aa', 'bb', 'cc'], ['dd', 'ee', 'ff'], ['hh', 'ii', 'jj']]
Lists of Lists

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

Lists, a data structure in programming, 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

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

```python
>>> y[0]
['aa', 'bb', 'cc']
```

```python
>>> y[1]
['dd', 'ee', 'ff']
```

```python
>>> y[2]
['hh', 'ii', 'jj']
```

*how do we access 'bb'?*
EXERCISE

```python
>>> 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
```