# Python Tutorial

- Python Data Types

- Python Control Structures

- Python Functions and Modules

1

# Temperature Converter: Example Program

- Problem Analysis
  - the temperature is given in Celsius, user wants it expressed in degrees Fahrenheit

- Program Specification
  - Input – temperature in Celsius
  - Output – temperature in Fahrenheit: 9/5(input) + 32

```python
>>> def main():
        celsius = eval(input("What is the Celsius temperature? "))
        fahrenheit = (9/5) * celsius + 32
        print("The temperature is ",fahrenheit," degrees Fahrenheit.")
```

```
>>> main()
What is the Celsius temperature? 0
The temperature is  32.0  degrees Fahrenheit.
>>> main()
What is the Celsius temperature? 100
The temperature is  212.0  degrees Fahrenheit.
```

# Python Syntax

- **Functions** are defined by "def" keyword

- No braces "{}" for scopes unlike C

- Purple keywords (highlighted in IDLE) are Python's **built-in functions**

  - Other examples include return

```python
def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = (9/5) * celsius + 32
    print("The temperature is ",fahrenheit," degrees Fahrenheit.")
```

User Input:    사용자가 화면에 입력한 값을 프로그램의 변수에 저장

### input의 사용

```
>>> a = input()
Life is too short, you need python
>>> a
'Life is too short, you need python'
>>>
```

input은 입력되는 모든 것을 문자열로 취급한다.

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요: 3
>>> print(number)
3
>>>
```

# Elements of Programs: Input Statements [2/2]

- The purpose of an input statement is to get input from the user and store it into a variable

$$\text{<variable>} = \text{eval(input(<prompt>))}$$

- First the prompt is printed
- The `input` part waits for the user to enter a value and press <enter>
- The expression that was entered is `eval`uated to turn it from a string of characters into a Python value (a number)
- The value is assigned to the variable

- >> x = input("type your value:")

- >> x = eval(input("type your value:"))

# Elements of Programs: Identifiers        [1/2]

- ## Names
    - Names are given to variables (celsius, fahrenheit), modules (main, convert), etc
    - These names are called **identifiers**
    - Every identifier must begin with a letter or underscore ("_"), followed by any sequence of letters, digits, or underscores
    - Identifiers are **case sensitive**

- ## These are all different, valid names
    - X              Celsius          fahrenheit
    - Spam                 spam               spAm
    - Spam_and_Eggs          Spam_And_Eggs

· Reserved Words

   · Some identifiers are part of Python itself. These identifiers are
      known as **reserved words or keywords**

   · This means they are not available for you to use as a name for a
      variable, etc. in your program

| and | del | for | is | raise |
|------|---------|--------|--------|--------|
| assert | elif | from | lambda | return |
| break | else | global | not | try |
| class | except | if | or | while |
| continue | exec | import | pass | yield |
| def | finally | in | print | |

Table 2.1: Python Reserved Words.

# Elements of Programs: Expressions

- The fragments of code that produce or calculate new data values are called expressions
- Literals are used to represent a specific value, e.g. 3.9 or 1 or 1.0
- Simple identifiers can also be expressions

```
>>> x = 5
>>> x
5
>>> print(x)
5
```

User Output:    프로그램의 변수를 화면에 출력하는 작업

```
>>> a = 123
>>> print(a)
123
>>> a = "Python"
>>> print(a)
Python
>>> a = [1, 2, 3]
>>> print(a)
[1, 2, 3]
```

### 큰따옴표(")로 둘러싸인 문자열은 + 연산과 동일하다

```
>>> print("life" "is" "too short") # ①
lifeistoo short
>>> print("life"+"is"+"too short") # ②
lifeistoo short
```

### 문자열 띄어쓰기는 콤마로 한다

```
>>> print("life", "is", "too short")
life is too short
```

```
>>> for i in range(10):
...     print(i, end=' ')
...
0 1 2 3 4 5 6 7 8 9
```

# Elements of Programs: Output Statements   [2/2]

- A print statement can print any number of expressions
- Successive print statements will display on separate lines
- A bare print will print a blank line

- `print(3+4)` ➔ `>> 7`
- `print(3, 4, 3+4)` ➔ `>> 3, 4, 7`
- `print()` ➔ `>>`
- `print("The answer: ", 3+4)` ➔ `>> The answer: 7`

Special Characters in String Data Types

"\n" → new line character (enter key)
"\t" → TAB character (tab key)

In IDLE , typing '\' shows

# Elements of Programs: Assignment Statements

- Simple Assignment
    - \<variable> = \<expr>
    - \<variable> is an identifier, \<expr> is an expression

- The expression on the RHS is evaluated to produce a value which is then associated with the variable named on the LHS

```
x = 3.9 * x * (1-x)
fahrenheit = 9/5 * celsius + 32
x = 5
```

# Elements of Programs: Simultaneous Assignment

- Several values can be calculated at the same time
    - \<var\>, \<var\>, … = \<expr\>, \<expr\>, …
    - Evaluate the expressions in the RHS and assign them to the variables on the LHS

```
sum, diff = x+y, x-y
```

- How could you use this to swap the values for x and y?
    - Would this work?
      x = y
      y = x

- We could use a temporary variable…

- Or We can swap the values of two variables quite easily in Python!

```
x, y = y, x
>>> x = 3
>>> y = 4
>>> print (x, y)
3 4
>>> x, y = y, x
>>> print (x, y)
4 3
```

- A *definite* loop executes a definite number of times
    - i.e., at the time Python starts the loop it knows exactly how many *iterations* to do

  for <var> in <sequence>:
      <body>

- The beginning and end of the body are indicated by **indentation**
- Examples

```
>>> for i in [0,1,2,3]:
        print (i)
```

```
0
1
2
3
```

```
>>> for odd in [1, 3, 5, 7]:
        print(odd*odd)
```

```
1
9
25
49

>>>
```

# Elements of Programs: Definite Loops [2/2]

- Python built-in function range() returns a range object
- Python built-in function list()  returns a list of things inside the paranthesis

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
for i in range(4):
```
*is equivalent to*
```
for i in [0, 1, 2, 3]:
```

```
>>> answer = []
    for i in range(1,10):
        answer.append(str(i))
```

참고: answer = answer.append(str(i)) 는 error!
왜냐하면 append는 answer에 append하는것이고 return은 없으므로

# Numeric Data Types

· Types (= classes)
  · Integers (int) – whole numbers
    · E.g. 3, 5
  · Floating point values (float) – with decimal point
    · E.g. 3.1, 5.1, 6.

· Types can be probed using "type()" built-in function

```
>>> type(3)
<class 'int'>
>>> type(3.1)
<class 'float'>
>>> type(3.0)
<class 'float'>
```

# Numeric Data Types: Operations

· Operations on ints produce ints, operations on floats produce floats (except for /).

```
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3.3333333333333335
>>> 10 // 3
3
>>> 10.0 // 3.0
3.0
>>> 3.0 + 4
7.0
>>> 3.0+4.0
7.0
>>> 3.0*4.0
12.0
>>> 3*4
12
```

# Boolean Data Type and Expression

· There is additional type called *bool* – it's either **True** or **False**

```
>> type(True)
<class 'bool'>
```

· Boolean expressions are always evaluated to **True** or **False**
· Format: `<expr> <rel-op> <expr>`

| Python | Mathematics | Meaning |
|--------|-------------|---------|
| < | < | Less than |
| <= | ≤ | Less than or equal to |
| == | = | Equal to |
| >= | ≥ | Greater than or equal to |
| > | > | Greater than |
| != | ≠ | Not equal to |

# Boolean Expressions: Comparisons by Types

· Operands in comparison operations should be compatible types

· However, test for equality can be carried out with different types (but returns "False")

· When comparing strings, **the ordering is lexicographic**

```
>> "aaa" > "abb"
False


>> 5 < 2.5
False


>> "a" < 572.0
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    "a" < 572.0
TypeError: unorderable types: str() < int()


>> "a" == 572.0
False
```

# Boolean Expressions:  Multi-operand comparison

· The following compound comparisons are valid expressions in Python
E.g.

```
>> 1 < 5 < 7
True


>> 2 > 1 < 7
True


>> 5 > 4 > 3.2 >= 1 == 1 != 8
True
```

```
If  0 <= number <= 100 :
     print(number)              # 이것도
가능
```

# Composite Boolean Expressions

- Boolean operators `and, or, not.`

    `<expr> and <expr>`

    `<expr> or <expr>`

    `not <expr>`

- We can represent their semantics (meaning) using a *truth table*

| P | Q | P and Q | P or Q | not Q |
|---|---|---------|--------|-------|
| T | T | T | T | F |
| T | F | F | T | T |
| F | T | F | T | - |
| F | F | F | F | - |

- The order of operators: not > and > or
- For example,
    - `a or not b and c`
    - `(a or ((not b) and c))`

# Boolean Algebra

- Anything `or`ed with true is true:
  ```
  a or true == true
  ```

- Both `and` and `or` are  distributive:
  ```
  a or (b and c) == (a or b) and (a or c)
  a and (b or c) == (a and b) or (a and c)
  ```

- Double negatives cancel out:
  ```
  not(not a) == a
  ```

- DeMorgan's laws:
  ```
  not(a or b) == (not a) and (not b)
  not(a and b) == (not a) or (not b)
  ```

# Boolean Expressions: Evaluating Other Types

- Python will let you evaluate **any built-in data type** as a Boolean

- For numbers (int, float), zero is considered **False**, anything else is considered **True**

- An empty sequence is considered as False while any non-empty sequence is taken to mean True

- The Boolean operators have operational definitions that make them useful for other purposes

```
>>> bool(0)                  >>> bool(1)
False                          True
>>> bool(32)                  >>> bool("Hello")
True                           True
>>> bool("")                 >>> bool([1,2,3])
False                          True
>>> bool([])
False
```

23

# Data Types과 연산

- Basic Data Types
  - Integer
  - Floating Number
  - Boolean
  - Character

우리가 익숙한 mathematical notation으로 연산

Ex: 3 + 4

- Advanced Data Types
  - Tuple
  - String
  - List
  - Dictionary
  - Set

특정 data type에 정의된 function들을 call해서 연산
   Ex:   myString = "S N U"
           myString.split()

- User-Defined Data Types (Classes)
  - Student
  - Automobile
  - ……

특정 data type에 정의된 function들을 call해서 연산
Ex: myAuto = Automobile("GM", "2016", "5Door")
           myAuto.

- Library
  - Math
  - Random
  - ….

특정 library에 정의된 function들을 call해서 연산
Ex:   import math
           math..sqrt(4)

24

# Factorial Program        [1/2]

```python
# factorial.py
#    Program to compute the factorial of a number
#    Illustrates for loop with an accumulator

def main():

    n = eval(input("Please enter a whole number: "))

    fact = 1
    for factor in range(n,1,-1):
        fact = fact * factor

    print("The factorial of", n, "is", fact)
```

```
>>> main()
Please enter a whole number: 100
The factorial of 100 is
  93326215443944152681699238856266700490715968264
  38162146859296389521759999322991560894146397615
  65182862536979208272237582511852109168640000000
  000000000000000
```

· Interesting thing to note is that Python **expands** integers into biginteger automatically

· Python has **built-in support** for integers exceeding 32-bit or 64-bit

# Type Conversions

```
>>> float(22//5)
4.0
>>> float(3)
3.0
>>> float(3.3)
3.3
>>> int(4.5)
4
>>> int(4.9)
4
>>> int(4.1)
4
```

```
>>> int(4.9999999999)
4
>>> int(4)
4
>>> round(3.9)
4
>>> round(3)
3
>>> round(3.5)
4
>>> str(8)
'8'
```

```
>>> 32/32
1.0
>>> 3/2
1.5
>>> 100.0000000
100.0
>>> 100.0
100.0
>>> 100
100
>>> 3/2.0
1.5
>>> 3/2
1.5
```

```
>>> 32//32
1
>>> 3//3
1
```

```
>>> 100.0000000
100.0
>>> 100.0
100.0
>>> 100
100
>>> 3/2.0
1.5
>>> float(3/2)
1.5
>>> float(3)/2
1.5
>>> float(5)/4
1.25
>>> int(3.0)
3
```

# Some Numeric Computations in Python 3.6   [2/2]

```
>>> 5 % 4
1
>>> 10 % 5
0
>>> 11 % 5
1
>>> 12 % 5
2
>>> float(5)/4
1.25
>>> int(3.0)
3
>>> 5 % 4
1
>>> 10 % 5
0
```

```
>>> 11 % 5
1
>>> 12 % 5
2
>>> 12**2
144
>>> 16**0.5
4.0
>>> 24**0.5
4.898979485566356
>>> 5 + (35 8 13 +2)
SyntaxError: invalid syntax
>>> 5 + (35 + 13 + 2)
55
>>> 5*(5+5)
50
```

| H | e | l | l | o |   | B | o | b |
|---|---|---|---|---|---|---|---|---|

0   1   2   3   4   5   6   7   8

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```

# String and Its Indexing [2/2]

| H | e | l | l | o |   | B | o | b |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

· In a string of *n* characters, the last character is at position *n-1* since we start counting with 0.

· We can index from the right side using negative indexes.
```
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```

# String and *Substring*

- Slicing:   <string>[<start>:<end>]
    - start and end should both be ints
    - contains the substring beginning at position start and runs up to **but doesn't include** the position end

| H | e | l | l | o |   | B | o | b |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```

```
>>> greet[0:-3]
'Hello '
>>> greet[:-1]
'Hello Bo'
```

# String and Its Operators

- Concatenation (+)
  >>"a" + "b"
  'ab'

- Repetition (*)
  >> "a" * 3
  'aaa'

- Length (len)
  >> len("a" * 3)
  3

# Program Example with String Operation

```
>>> Main()
     # get user's first and last names
     first = input("Please enter your first name (all lowercase): ")
     last  = input("Please enter your last name (all lowercase): ")

     # concatenate first initial with 7 chars of last name
     print ("your_name = ", first[0] + ". " + last[:7])
```

```
>>> main()
>>> Please enter your first name (all lowercase): john
>>> Please enter your last name (all lowercase): doe
>>> your_name =  j. doe
```

# Character to Numeric Conversion

- ord() : Python built-in function which returns the numeric (ordinal) code of a single character

- chr() : Python built-in function which converts a numeric code to the corresponding character

```
>>> ord("A")    # argument character의 asci code에서의 위치
65
>>> ord("a")
97
>>> chr(97)     # 0 ~~ 255를 받아서 ascii code를 return
'a'
>>> chr(65)
'A'
```

UTF-8 code (가변형 4 bytes)을 사용했다 하더라도 처음 1byte는 ASCII code와 동일함

Character Encoding

| ASCII | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | $N_U$ | $S_H$ | $S_X$ | $E_X$ | $E_T$ | $E_Q$ | $A_K$ | $B_L$ | $B_S$ | $H_T$ | $L_F$ | $Y_T$ | $F_F$ | $C_R$ | $S_0$ | $S_I$ |
| 0001 | $D_L$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $N_K$ | $S_Y$ | $E_\Sigma$ | $C_N$ | $E_M$ | $S_B$ | $E_C$ | $F_S$ | $G_S$ | $R_S$ | $U_S$ |
| 0010 |  | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | − | . | / |
| 0011 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 0100 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 0101 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 0110 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 0111 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | $D_T$ |
| 1000 | $8_0$ | $8_1$ | $8_2$ | $8_3$ | $I_N$ | $N_L$ | $S_S$ | $E_S$ | $H_S$ | $H_J$ | $Y_S$ | $P_D$ | $P_V$ | $R_I$ | $S_2$ | $S_3$ |
| 1001 | $D_C$ | $P_1$ | $P_Z$ | $S_E$ | $C_C$ | $M_M$ | $S_P$ | $E_P$ | $Q_8$ | $Q_Q$ | $Q_A$ | $C_S$ | $S_T$ | $O_S$ | $P_M$ | $A_P$ |
| 1010 | $A_0$ | ¡ | ¢ | £ | ¤ | ¥ | ¦ | § | ¨ | © | ª | « | ¬ | | ® | ‾ |
| 1011 | ° | ± | ² | ³ | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ | ¾ | ¿ |
| 1100 | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï |
| 1101 | Ð | Ñ | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| 1110 | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï |
| 1111 | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù | ú | û | ü | ý | þ | ÿ |

Figure 7.3   ASCII, the American Standard Code for Information Interchange.

*Note*: The original 7-bit ASCII is the top half of the table; the whole table is known as Extended ASCII (ISO-8859-1). The 8-bit symbol for a letter is the four row bits followed by the four column bits (e.g., A = 0100 0001, while z = 0111 1010). Characters shown as two small letters are control symbols used to encode nonprintable information (e.g., $B_S$ = 0000 1000 is backspace). The bottom half of the table represents characters needed by Western European languages, such as Icelandic's eth ( ð ) and thorn ( Þ ).

لماذا لا يتكلمون اللّغة العربية فحسب؟
Защо те просто не могат да говорят **български**
Per què no poden simplement parlar en **català**?
他們爲什麽不說中文（台灣）？
Proč prostě nemluví **česky**?
Hvorfor kan de ikke bare tale **dansk**?
Warum sprechen sie nicht einfach **Deutsch**?
Μα γιατί δεν μπορούν να μιλήσουν **Ελληνικά**;
Why can't they just speak **English**?
¿Por qué no pueden simplemente hablar en **castella**
Miksi he eivät yksinkertaisesti puhu **suomea**?
Pourquoi, tout simplement, ne parlent-ils pas **frança**
למה הם פשוט לא מדברים **עברית**?
Miért nem beszélnek egyszerűen **magyarul**?
Af hverju geta þeir ekki bara talað **íslensku**?
Perché non possono semplicemente parlare **italian**
なぜ、みんな日本語を話してくれないのか
의 모든 사람들이 한국어를 이해한다면 얼마나
Waarom spreken ze niet gewoon **Nederlands**?
Hvorfor kan de ikke bare snakke **norsk**?
Dlaczego oni po prostu nie mówią po **polsku**?
Porque é que eles não falam em **Português (do Br**
Oare ăştia de ce nu vorbesc **româneşte**?
Почему же они не говорят **по-русски**?
Zašto jednostavno ne govore **hrvatski**?
Pse nuk duan të flasin vetëm **shqip**?
Varför pratar dom inte bara **svenska**?
ทำไมเขาถึงไม่พูด**ภาษาไทย**
Neden **Türkçe** konuşamıyorlar?

re 7.4  "Why can't they just speak _____?"
Web page, www.trigeminal.com/samples/pro
displaying that question expressed in more th
ages. Can you name all of them in this partial

# Representations of Characters   [2/2]

- **ASCII:   7 bits for 128 characters  (1960년)**

- **E-ASCII: 8 bits for 256 characters  (1963년)**

- **유니코드**(Unicode)
  - 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현되도록 설계된 산업 표준
  - 유니코드 협회(Unicode Consortium)

- **UTF** (Unicode Transformation Format) **-8** :  32bits   (1992년)
  - A character encoding for all possible characters in Unicode
  - variable-length (➔ uses *1 to 4  units of 8-bit code)*
  - *UTF-8의  unit  1 ➔  E-ASCII 문자*
  - the dominant character encoding for the World Wide Web, accounting for 84.6% of all Web pages

# String Splitting Function

- How do we get the sequence of numbers to decode?
  - Read the input as a single string, then split it apart into substrings, each of which represents one number
- The string class has a set of methods
- split() : a function in string class
  - splits the given string into substrings based on spaces.

```
>>> a = "Hello string methods!"
```

```
>>> a.split()
['Hello', 'string', 'methods!']
>>> "Hello string methods!".split()
['Hello', 'string', 'methods!']
```

a = a.split() 를  써도 OK,   a.split()은 list를 return하므로!
But,  a  = a.append("dd")를 하면  a에는 nothing !

# String Splitting function

· Split can be used on characters other than space, by supplying the character as a parameter

```
>>> "32,24,25,57".split(",")
['32', '24', '25', '57']
>>> "abcPdefPghi".split("P")
['abc', 'def', 'ghi']


>>> list("CMU")
["C", "M", "U"]
```

# String functions belonging to "string" class

- s.capitalize()
- s.title()
- s.center(width)
- s.count(sub) – Count # of occurrences of sub in s
- s.find(sub) – Find first pos where sub occurs in s
- s.join(list) – Concatenate list of strings into one string using s as separator
- s.ljust(width)
- s.lower() – Copy of s in all lowercase letters
- s.lstrip() – Copy of s with leading whitespace removed
- s.replace(oldsub, newsub) – Replace occurrences of oldsub in s with newsub
- s.rfind(sub) – Like find, but returns the right-most position
- s.rjust(width) – Like ljust
- s.rstrip() – Like lstrip
- s.split()
- s.upper() – Like lower

```
---
>>> x = 'ham'
>>> x
'ham'
>>> x = "hamsandwich"
>>> x
'hamsandwich'
>>> y = x + "book"
>>> y
'hamsandwichbook'
>>> y = x + " book"
>>> y
'hamsandwich book'
```

```
>>> x = "hamsandwich"
>>> z = 10
>>> y = x + z
Traceback (most recent call last):
  File "<pyshell#95>", line 1, in <module>
    y = x + z
TypeError: must be str, not int
>>> y = x + str(z)
>>> y
'hamsandwich10'
>>> y = "something %d" %z
>>> y
'something 10'
>>> y = "something %f" %z
>>> y
'something 10.000000'
>>> y = "something %.3f" %z
>>> y
'something 10.000'
>>> z = 1.6546546548
>>> y = "something %.3f" %z
>>> y
'something 1.655'
```

```
>>> "ham" in "hamsandwich"
True
>>> 'a' in 'ham'
True
>>> x = []
>>> x
[]
>>> x = ["ham", 4, 2.2]
>>> x
['ham', 4, 2.2]
>>> x.append(5)
>>> x
['ham', 4, 2.2, 5]
>>> x.insert(1,3.1415)
>>> x
['ham', 3.1415, 4, 2.2, 5]
```

```
>>> x.pop(1)
3.1415
>>> x
['ham', 4, 2.2, 5]
>>> len("words")
5
>>> len(x)
4
```

```
>> x = ['a', 'b', 'c']                    >> x = ['a', 'b', 'c']
>> x = x.pop(1)            >> x.pop(1)
>> x                       >> x
```

```
>>> list("ham")
['h', 'a', 'm']
>>> x = "ham"
>>> y = list(x)
>>> y
['h', 'a', 'm']
>>> y.append(x)
>>> y
['h', 'a', 'm', 'ham']
>>> list("ham")
['h', 'a', 'm']
>>> y=[]
>>> y.append("ham")
>>> y
['ham']
```

```
>>> "s"
's'
>>> "s" in "something"
True
>>> "s" in y
False
```

```
y = y.append(x)    # 절대 안됨
y = list(y)        # 가능함
```

- Lists are a special kind of *sequence*, so sequence operations also apply to lists!

```
>>> [1,2] + [3,4]
[1, 2, 3, 4]

>>> [1,2]*3
[1, 2, 1, 2, 1, 2]

>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
>>> len(grades)
5
```

# List Data Type        [2/2]

· Strings are always sequences of characters
    univ_name = "Seoul National Univ"

· but lists can be sequences of arbitrary values
    · Lists can have numbers, strings, or both!

      myList = [1, "Spam ", 4, "U"]

      myList = [3,  9,  1,   7]

# Program Example with List data type

```python
# month2.py
#  A program to print the month name, given it's number.
#  This version uses a list as a lookup table.

def main():

    # months is a list used as a lookup table
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

    n = eval(input("Enter a month number (1-12): "))

    print ("The month abbreviation is", months[n-1] + ".")
```

# Mutability Issue of List and String Data Type

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
```

*mutable* (i.e. they can be

```
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = "p"     #This is not allowed
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    myString[2] = "p"
TypeError: object doesn't support item assignment

>>> myString = "Hi World"      #This is OK
```

immutable
: Parts of
Strings can
not be
changed
using

47

# Tuple Data Type

```
>>> t1 = ()
>>> t2 = (1, )
>>> t3 = (1, 2, 3]
>>> t4 = ['a', 'b',   ['ab',
'cd']
```

- Tuple is similar to list
  But, Tuple is *immutable*
  *Tuple is more memory efficient than List*

```
>>> t1 = [ ]
>>> t2 = [1,]
>>> t3 = [1,2,3]
>>> t4 = ['a', 'b',   ['ab',
'cd']
```

```
>>> x = ("ham",4,5)
>>> x
('ham', 4, 5)
>>>
```
inside a tuple!

```
>>> x[2] = 8    # not
allowed
```

Whole replacement is fine!

```
>>> x = ("egg", 7, 9, 10)
```

# Set Data Type

```
>>> s1 = { 1,   2, 3}
>>> s2 = {"ab", "cd"}
```

Built-in set() function을 써서 set data를 만들수 있다.   Set()의 parameter는 1개만 들어가야 한다

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
```

```
>>> s2 = set("Hello")
>>> s2
{'e', 'l', 'o', 'H'}
```

```
>>> s1 = set([1,2,3])
>>> l1 = list(s1)
>>> l1
[1, 2, 3]
>>> l1[0]
1
```

```
>>> s1 = set([1,2,3])
>>> t1 = tuple(s1)
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

# Set Operations     [1/2]

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

```
>>> s1 & s2
{4, 5, 6}
```
=
```
>>> s1.intersection(s2)
{4, 5, 6}
```

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```
=
```
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
```
=
```
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

# Set Operations　　[2/2]

## 값 1개 추가하기(add)

이미 만들어진 set 자료형에 값을 추가할 수 있다.
다.

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

## 값 여러 개 추가하기(update)

여러 개의 값을 한꺼번에 추가(update)할 때

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

## 특정 값 제거하기(remove)

특정 값을 제거하고 싶을 때는 아래의

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

# Dictionary Data Type

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

| key | value |
|-----|-------|
| name | pey |
| phone | 01199993323 |
| birth | 1118 |

```
>>> dic['name']
'pey'
>>> dic['phone']
'0119993323'
>>> dic['birth']
'1118'
```

# Insert and Delete in Dictionary

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{2: 'b', 1: 'a'}
```

```
>>> a['name'] = 'pey'
{'name':'pey', 2: 'b', 1: 'a'}
```

```
>>> a[3] = [1,2,3]
{'name': 'pey', 3: [1, 2, 3], 2: 'b', 1: 'a'}
```

```
>>> del a[1]
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

# Dictionary 만들 때 주의사항

중복되는 Key 값은  금지

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

Key 값은  immutable value만 허락

```
>>> a = {[1,2] : 'hi'}
Traceback (most recent call last):
File "", line 1, in ?
TypeError: unhashable type
```

# Python Tutorial

· Python Data Types

· Python Control Structures

· Python Functions and Modules

# One-way Decision Structures

```
if <condition>:
    <statements>
```

```
if <condition>:
    <statements>
else:
    <statements>
```
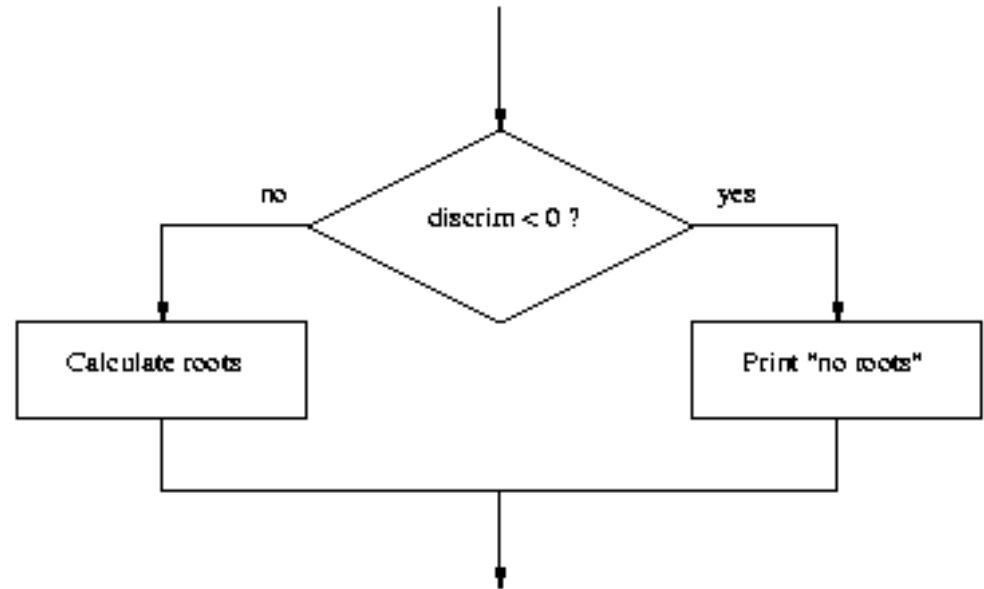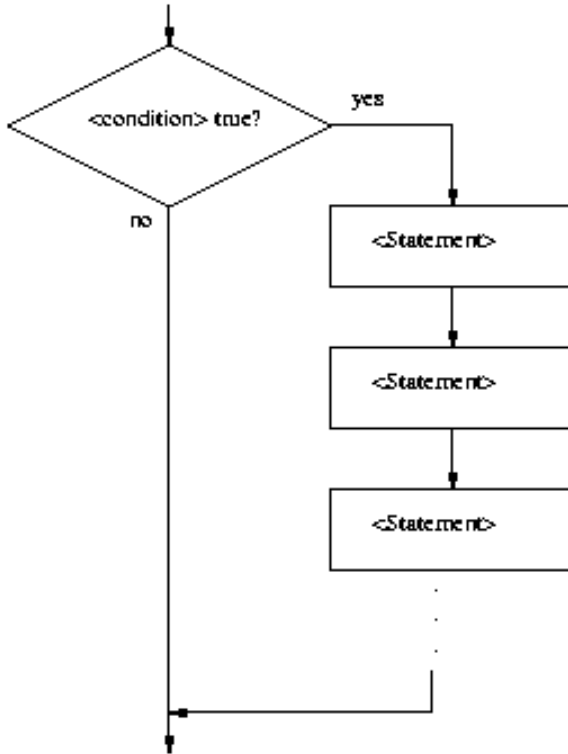
# Some If statements in Python 3.6

```python
>>> mail = 0
>>> if mail:
        print('mail time')
else:
        print('no mail:(')


no mail:(
>>>
```

```python
>>> if(7 <=6):
        print("whaaaa")
else:
        print("7 is GREATER than 6")


7 is GREATER than 6
>>> if (7) and (6):
        print('yep')


yep
>>> if (0) and (4):
        print('wahaaa')


>>> if not(0):
        print('yep')


yep
>>>
```

# One-way Decision Structure: Example

```python
# convert2.py
#      A program to convert Celsius temps to Fahrenheit.
#      This version issues heat and cold warnings.

def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = 9 / 5 * celsius + 32
    print("The temperature is", fahrenheit, "degrees fahrenheit.")
    if fahrenheit >= 90:
        print("It's really hot out there, be careful!")
    if fahrenheit <= 30:
        print("Brrrrr. Be sure to dress warmly")
```

# Multi-Way Decision Structures

```
if <condition>:
    <statements>
elif <condition>:
    <statements>

else:
    <statement
```
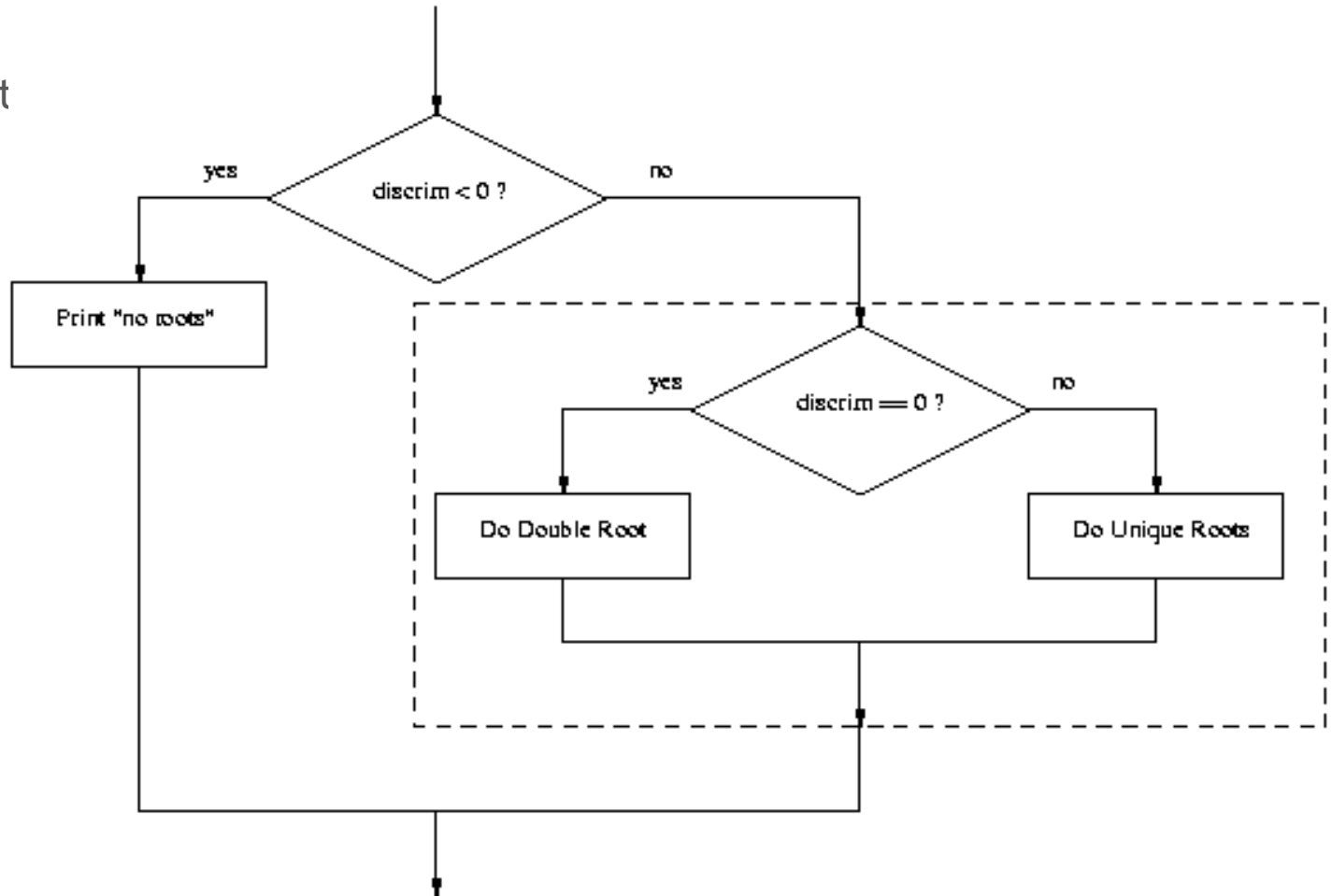
# One-way Decision Structure: Example

- Temperature Warnings
  - Let's say we want to modify that program to print a warning when the weather is **extreme**



```
Input Celsius Temperature
Farenheit = 9/5 * celsius + 32
Print Fahrenheit
```

fahrenheit > 90?  →  yes  →  Print a Heat Warning

no

fahrenheit < 30?  →  yes  →  Print a Cold Warning

no
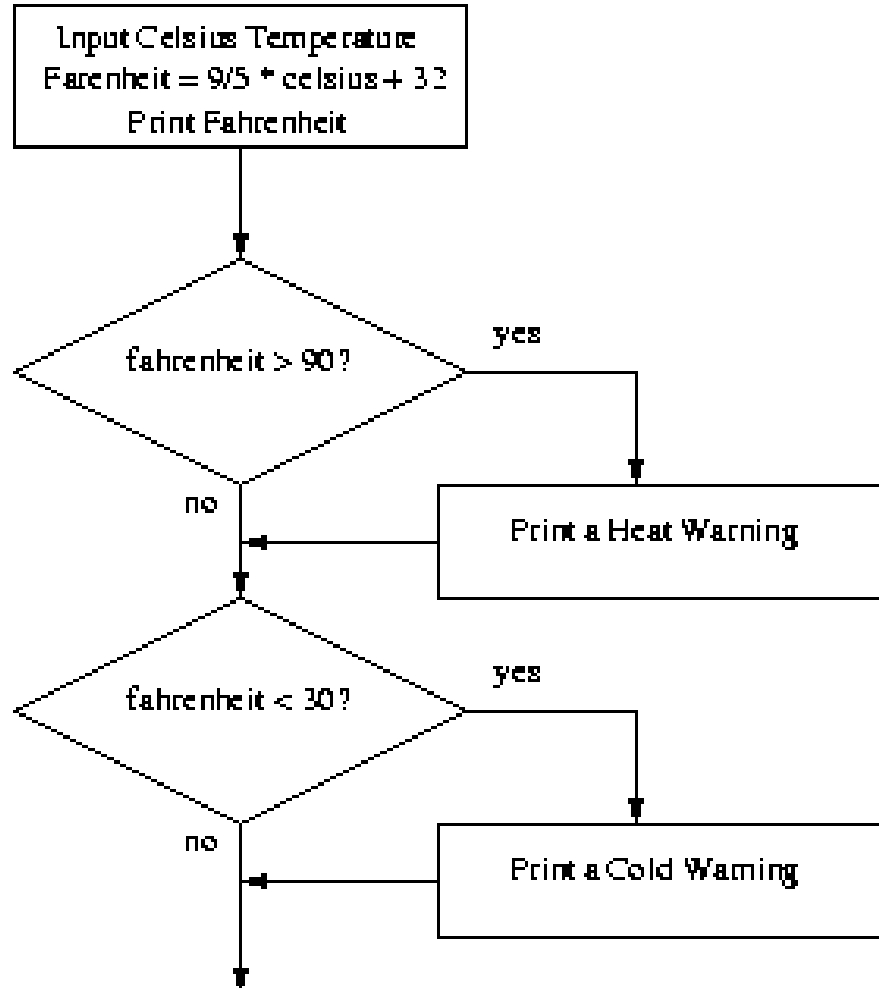
# Two-Way Decision Structure: Example

```python
# quadratic3.py

#    A program that computes the real roots of a quadratic equation.

#    Illustrates use of a two-way decision


import math

def main():
    print "This program finds the real solutions to a quadratic\n"
    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))


  discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print ("\nThe solutions are:", root1, root2 )
```

# Multi-Way Decision Structure: Example

```python
# quadratic4.py
#     Illustrates use of a multi-way decision

import math

def main():
    print("This program finds the real solutions to a quadratic\n")
    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
```
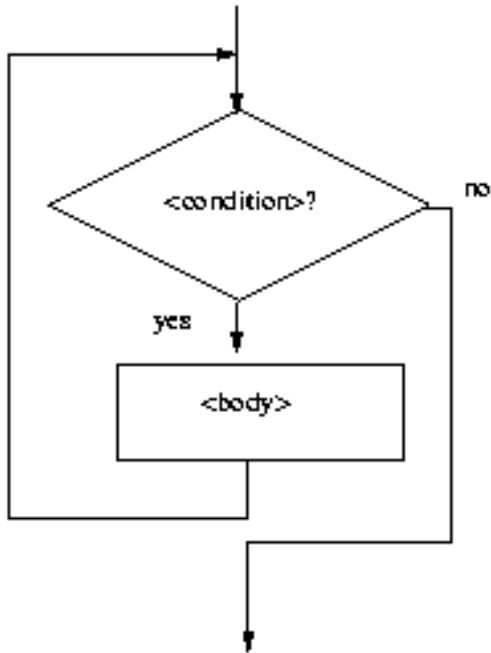
# Loop Structures

· Loop types classified by **breaking condition**

· **Interactive Loop**: user dictates whether to continue the loop interactively
  · **Sentinel Loop**: loop is carried on until certain condition is met
  · **End-of-file Loop:** looped until end of the file
  · **Nested Loop:** loop in a loop

· Loop structures
  · For loop
  · While loop

# Loop Structures

```
while <condition>:
    <loop body>
```



- a while loop that counts from 0 to 10:

```
i = 0
while i <= 10:
    print(i)
    i = i + 1
```

- for loop that has the same output

```
for i in range(11):
    print(i)
```

# Loop statements in Python 3.6 [1/2]

```
>>> x = [1,2,7]
>>> for i in x:
        print(i)


1
2
7
>>>
...
```

```
>>> for i in range(30):
        print(i)


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
>>>
```

# Loop statements in Python 3.6 [2/2]

```
>>> for i in range(10,30,2):
        print(i)

10
12
14
16
18
20
22
24
26
28
>>>
```

```
>>> for i in range(30):
        if not (i % 3):
                continue
        print(i)

1
2
4
5
7
8
10
11
13
14
16
17
19
20
22
23
25
26
28
29
>>>
```

## 'WHILE' LOOP

while (condition is true):
  do this over and over

x = 0
while (x < 10):
  x += 1

## 'BREAK' LOOP

- Used to STOP loop

while (true):
  if (something):
    break

```python
>>> x,y = 0,0
>>> while(True):
        x += 1
        y += 2
        if(x + y > 10):
            break

>>> x
4
>>> y
8
```

# Indefinite Loops: Warning

· The while statement is simple and powerful, could be **dangerous**!

```
i = 0
while i <= 10:
    print(i)
```

· It can be easy to omit the incrementing logic (applies to all languages)

# Interactive Loops: Average computation example

· **Basic pseudocode**

```
set moredata to "yes"
while moredata is "yes"
    get the next data item
    process the item
    ask user if there is moredata
```

· **Example**
```
def main():
    moredata = "yes"
    sum = 0.0
    count = 0
    while moredata[0] == 'y':
        x = eval(input("Enter a number: "))
        sum = sum + x
        count = count + 1
        moredata = input("More numbers (yes or no)? ")
    print("\nThe average of the numbers is", sum / count)
```

## Interactive Loops: Example

```
Enter a number: 32
Do you have more numbers (yes or no)? y
Enter a number: 45
Do you have more numbers (yes or no)? yes
Enter a number: 34
Do you have more numbers (yes or no)? yup
Enter a number: 76
Do you have more numbers (yes or no)? y
Enter a number: 45
Do you have more numbers (yes or no)? nah

The average of the numbers is 46.4
```

# Sentinel Loops

· Continues to process data until reaching a special value (called the **sentinel 보초, 파수병**) that signals the end

· The sentinel must be distinguishable from the data
  · since it is not processed as part of the data

```python
# average3.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using negative input as sentinel

def main():
    sum = 0.0
    count = 0
    x = eval(input("Enter a number (negative to quit) >> "))
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = eval(input("Enter a number (negative to quit) >> "))
    print("\n The average of the numbers is", sum / count)
```

➢ Assuming there are no negative numbers in the data

71

# Sentinel Loops: Using empty string as the sentinel

```python
# average4.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using empty string as sentinel

def main():
    sum = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")

    print("\n The average of the numbers is", sum / count)
```

# Loop with readline() through a File: Example

```python
# average6.py
#      Computes the average of numbers listed in a file.

def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        sum = sum + eval(line)
        count = count + 1
        line = infile.readline()

    print("\n The average of the numbers is", sum / count)
```

# Nested Loops in reading lines through a file: Example

- We want to read any number of numbers on a line in the file (separated by commas)

  3, 4, 5, 6, 1, 2, …, 1

  3, 2, 1, 7, 5, 2, …, 1

  5, 6, 4, 7, 5, 6, …,

- We use two loops:
  - The top-level loop loops through each line of the file
  - The second-level loop loops through each number of each line

```python
# average7.py
#      Computes the average of numbers listed in a file.
#      Works with multiple numbers on a line.

import string
def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split(","):
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
    print("\n The average of the numbers is", sum / count)
```
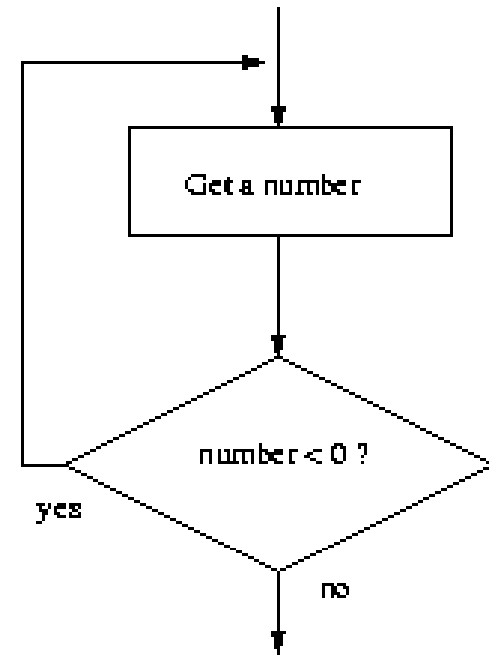
74

# Post-Test Loops

- Condition test comes after the body of the loop

  - A post-test loop always executes the body of the code at

    least once

- Python doesn't have a built-in statement to do this,

  - but we can do it with a slightly modified **while** loop.

  - In some other languages (not in Python);

    *repeat*

    *get a number from the user*

    *until number >= 0*

# Post-Test Loops: Example

- Using a `while` statement
  - Seed the loop condition so we're guaranteed to execute the loop once.

```python
number = -1
while number < 0:
    number = eval(input("Enter a positive number: "))
```

  - By setting `number` to –1, we force the loop body to execute at least once

- The same algorithm implemented with a `break`:

```python
while True:
    number = eval(input("Enter a positive number: "))
    if x >= 0: break
```

  - Executing `break` causes Python to immediately exit the enclosing loop

# File Opening and File Closing

- *Opening* a file => associating a file on disk with an object in memory
  - **<filevar> = open(<name>, <mode>)**
  - Associate a disk file <name> with a file object <filevar>
  - <mode> is either 'r' or 'w'

    myfile = open("numbers.dat", "r")
  - We can manipulate the file by manipulating the file object "myfile"

- *Closing* the file causes any running operations and other bookkeeping for the file to be completed
  - In some cases, not properly closing a file could result in data loss

    myfile.close()

# Looping readline() through Files

· readline() function  can be used to read one line at a time

```
myfile = open("someFile.txt", "r")
for i in range(5):
  line = myfile.readline()
  print line[:-1]
```

# reads the first 5 lines of a file

# Slicing used to strip out the newline char at end of lines

# Writing a File

- outfile = open("mydata.out", "w")
  - Opening a file for writing prepares the file to receive data

- If file already exists
  - file's contents are erased (& starts with empty file).

- If file does not exist
  - new file is created

- outfile.write("new sample contents \n")

# Python Tutorial

- Python Data Types

- Python Control Structures

- Python Functions and Modules

# Taxonomy of Python Functions

- Python Built-in Functions:
  - input(), print(), len(), abs(), set(), …..

- User Defined Functions

- Functions belonging to Python Basic Data Types
  - >>> L = [3, 5, 5]          >>> S = {3, 5, 6}
  - >>> L.append(4)            >>> S.remove(5)

- Functions belonging to User-Defined Classes

  >>> class Box(object):              >>> bbb = Box()

          def  calc_space(self):             >>> space_value = bbb.calc_space()

                  ……

- Functions from Other Modules (consisting of Functions and Classes)
  - >>> import sam_module

- Functions from Python Standard Library (consisting of Functions and Classes)
  - >>> import math

# Python Built-in Functions

| | | Built-in Functions | | |
|---|---|---|---|---|
| abs() | dict() | help() | min() | setattr() |
| all() | dir() | hex() | next() | slice() |
| any() | divmod() | id() | object() | sorted() |
| ascii() | enumerate() | input() | oct() | staticmethod() |
| bin() | eval() | int() | open() | str() |
| bool() | exec() | isinstance() | ord() | sum() |
| bytearray() | filter() | issubclass() | pow() | super() |
| bytes() | float() | iter() | print() | tuple() |
| callable() | format() | len() | property() | type() |
| chr() | frozenset() | list() | range() | vars() |
| classmethod() | getattr() | locals() | repr() | zip() |
| compile() | globals() | map() | reversed() | __import__() |
| complex() | hasattr() | max() | round() | |
| delattr() | hash() | memoryview() | set() | |

# Using Libraries (or Modules) [1/3]

- Modules contain a set of useful functions or classes

- Some additional Python standard libraries like "math", "time" or "datetime" etc. can be imported using keyword "import"

- Python standard libraries are listed here: https://docs.python.org/3/library/
  - # Importing math standard library
  - >> import math
  - >> math.sqrt(3)
  - 1.7320508075688772
  - >> - b + math.sqrt(b * b – 4 * a * c) / ( 2 * a )

- help(<module name>) can give you lots of information

# Using Libraries (or Modules)       [2/3]

- When a Python program starts, it only has access to basic functions and classes

  (len(), sum(), set(), list(), range(), ….. )

- Use "import" to tell Python to load a module
  - "import" vs "from ... import ..."

```
>>> # If you want to use cos function in the math library
>>> import math
math.cos
>>> from math import cos, pi
cos
>>> from math import *
cos
```

```
>>> import math

>>> math.pi
3.1415926535897931
>>> math.cos(0)
1.0
>>> math.cos(math.pi)
-1.0


>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'exp',
 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log',
'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> help(math)
…….
>>> help(math.cos)
……..
```

# Creating Modules (Library): A Short Introduction

- Modules can be created very easily
  - Any script file will be considered a module if it is imported by another script file
- For example,
  - A script file named **hello.py** has the following line

  ```
  def sample_func()
      print("Hello World")
  ```

  - Another script file in the same directory can import **hello.py** by simply referring to its file name

  ```
  import hello
  Hello.sample_func()
  ```

- When a script file is imported like a module, all its defined **functions** and **classes** will be available to the importer