SI100B/F: Introduction to Information Science and Technology

Part I: Python Programming

邱越

School of Information Science and Technology

ShanghaiTech University

Basic Data Types and Assignments

1. Scalars

- scalar objects are indivisible
- have a single value
  - numeric types

    - `int`, `float`, `complex`
  - bool
    - `True`, `False`

2. Text Sequence Type

- strings
  - sequence of characters
  - `str`

3. Sequence Types

- `list`, `tuple`

4. Mapping Types

- `dict`

5. Sets

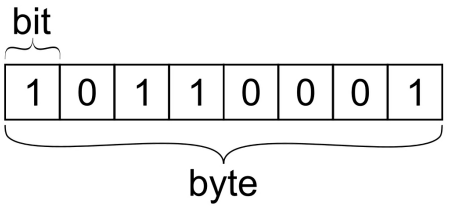
6. Assignment Operations

Basics About Memory

Basic units

- `bit`
  - short for binary digit
  - basic unit of information
  - either `0` or `1`
- `byte`
  - consist of 8 sequential bits

- basic unit of computer memory
- smallest addressable unit of memory



Rules

	SI		Binary
1 KB (kilobyte)	10^3 bytes	1 KiB(kibibyte)	2^{10} bytes
1 MB (megabyte)	10^3 KB	1 MiB(mebibyte)	2^{10} KiB
1 GB (gigabyte)	10^3 MB	1 GiB (gibibyte)	2^{10} MiB
1 TB (terabyte)	10^3 GB	1 TiB (tebibyte)	2^{10} GiB

SI: International System of Units

Variables

variable

- is an identifier/name used to refer a particular data/object stored in memory
- can only point to one specific object value at one time
- can vary and refer to different objects from time to time
- data/object can be retrieved using the variable "name"

naming schemes

- can only be composed of letters (a-z, A-Z), numbers (0-9), or underscore "_"
- must start with a letter or "_"
  - e.g., *name* and name are allowed, 3_name is not
- case sensitive

  - Name ≠ name
- reserved keywords can not be used to name a variable
- '=' is used to assign a variable

In [1]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'els
e', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'rai
se', 'return', 'try', 'while', 'with', 'yield']
```

Scalars: Python as a calculator

int

- represents a datum of integral data type
- only `signed int` type in Python
  - different with C/C++
- `int` variable takes 32 bits for 32 bit systems and 64 bits for 64 bit systems
  - positive: leftmost 0
  - negative: leftmost 1

float

- like scientific notation 1.23×10^{23}, but denoted as `1.23e23`
- stored with three parts
  - a sign: `+` or `-`
  - a mantissa: e.g., `1.23`

    - an exponent: e.g., `23`
- typically takes 64 bits/8 bytes

complex

- `<real part> + <imaginary part>j`
- `<real part> + <imaginary part>J`
- $j^2 = J^2 = -1$

In Python, `type()` returns the type of a variable

In [2]:
```
3 + 2
```

```
5
```

In [3]:
```
x = 5.0
type(x)
```

```
float
```

In [4]:
```
type(5)
```

```
int
```

In [5]:
```
3.0 + 2.0
```

```
5.0
```

In [6]:
```
(3 + 2j)*(1 - 2j)
```

```
(7-4j)
```

In [7]:
```
type((7-4j))
```

```
complex
```

In [8]:
```
(1+2j)/(3+4j)
```

```
(0.44+0.08j)
```

Operators on int and float

operator	description	syntax
+	addition	x + y
-	subtration	x - y
*	multiplication	x * y
/	division (return a float)	x / y
//	division (return an int)	x // y

operator	description	syntax
%	modulus	`x % y`
**	power x^y	`x ** y`

In [9]:
```
4/3
```

```
1.3333333333333333
```

In [10]:
```
5//3
```

```
1
```

In [11]:
```
5%3
```

```
2
```

Built-in Math Functions

operator	description	syntax
abs()	| x |	`abs(x)`
divmod()	queotient and remainder	`divmod(x, y)`
*	multiplication	`x * y`
/	division (return a float)	`x / y`

operator	description	syntax
//	division (return an int)	x // y
%	modulus	x % y
**	power x^y	x ** y

In [12]:
```
pow(2, 3)
```

```
8
```

In [13]:
```
abs(-5)
```

```
5
```

In [14]:
```
divmod(5, 3)
```

```
(1, 2)
```

In [15]:
```
pow(3, 2)
```

```
9
```

In [16]:
```
3**2
```

```
9
```

In [17]:
```
round(4/3, 5)
```

```
1.33333
```

In [18]:
```
help(round)
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.
```

Bool Type

- has only two possible values
  - `True` or `False`
  - the `bool()` function allows you to evaluate any value, and give you `True` or `False` in return

```
bool(7)
bool(1.2)
bool(1 + 2j)
bool(0)
bool(0.0)
bool(0+0j)
```

  - primarily associated with conditional statements (`if/while`)
  - any nonzero scalars, nonempty ones of list, tupe, set, dict are `True`

In [19]:
```
bool(1)
```

```
True
```

In [20]:
```
bool(1.2)
```

```
True
```

In [21]:
```
bool(1+2j)
```

```
True
```

In [22]:
```
bool(0)
```

```
False
```

In [23]:
```
bool(0.0)
```

```
False
```

In [24]:
```
bool(0+0j)
```

```
False
```

In [25]:
```
bool('abc')
```

```
True
```

In [26]:
```
bool('')
```

```
False
```

Relation Operations

Compares the values and returns bool type

operator	description	syntax
>	greater than	x > y
<	less than	x < y
==	equal to	x == y
!=	not equal to	x != y
>=	greater than or equal to	x >= y
<=	less than or equal to	x <= y

In [27]:
```
5 > 3
```

```
True
```

In [28]:
```
5 < 3
```

```
False
```

In [29]:
```
5 == 3
```

```
False
```

In [30]:
```
5 != 3
```

```
True
```

In [31]:
```
5 >= 3
```

```
True
```

In [32]:
```
5 <= 3
```

```
False
```

Logic Operators

Used to perform logic operations: and, or, not

x	y	x and y	x or y	not x
True	True	True	True	False

x	y	x and y	x or y	not x
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

In [33]:
```
5 > 3 and 4 > 2
```

```
True
```

In [34]:
```
5 > 3 or 3 > 5
```

```
True
```

In [35]:
```
not 5 > 3
```

```
False
```

Strings

- sequences of character data
- string type in Python is called `str`
- enclosed in single quotes '.....' or double quotes "....."
- escape characters for format control (Lecture 1)

```
print('don\'t do this!')
```

- using raw strings by adding `r` before the first quote without interpretation of escape characters

```
print(r'don\'t do this!')
```

- using tripple-quotes """....."""", '''.....''' for multiple lines of string without `\n`

In [36]:
```
print('don\'t do this!')
```

```
don't do this!
```

In [37]:
```
print(r'don\'t do this!')
```

```
don\'t do this!
```

In [38]:
```
print("""
 Name
     ls    - list directory contents
 Synopsis
     ls    [option] ... [file] ...
 Description
     -a    --all
 """)
```

```
 Name
     ls    - list directory contents
 Synopsis
     ls    [option] ... [file] ...
 Description
     -a    --all
```

In [39]:
```
print("
Name
    ls    - list directory contents
Synopsis
    ls    [option] ... [file] ...
Description
    -a    --all
")
```

```
  File "/tmp/ipykernel_628986/2271403094.py", line 1
    print("
          ^
SyntaxError: EOL while scanning string literal
```

String Operations

- concatination using +

```
'Py' + 'thon'
```

- repeatition using *

```
3 * 'Python'
```

- indexed using [pos] , pos starts from 0 at the beginning of the string

```
P  |  y  |  t  |  h  |  o  |  n
0     1     2     3     4     5
-6   -5    -4    -3    -2    -1
```

  - `word = 'Python'`
  - `word[0]` returns `P`
  - `word[0 : 2]` returns `Py`
  - `word[ : 2]` returns `Py`
  - indices `[a : b]` means `[a, b)`
- strings can also be indexed from the end using negative indices
  - `word[-1]` returns `n`

In [40]:
```
'Py' + 'thon' + ' is' + ' fun'
```

```
'Python is fun'
```

In [41]:
```
3*'Python'
```

```
'PythonPythonPython'
```

String Methods

Some widely used built-in string methods

- `strip()`
  - removes the whitespace at the beginning or the end

```
wd = 'his name '
wd.strip()
```

- `len()`
  - returns the length of the string

```
wd = 'Python'
len(wd)
```

- `lower()`
  - returns a string with lower case
- `upper()`
  - returns a string with upper case
- `replace(old, new)`
  - return a copy of the string with all occurrences of substring old replaced by new For details of string methods, cf. Python Official Site (https://docs.python.org/3/library/stdtypes.html#string-methods)

In [42]:
```
wd = 'Python'
len(wd)
```

```
6
```

In [43]:
```
wd = 'Python '
len(wd)
```

```
7
```

In [44]:
```
wd = 'Python'
print(wd.lower())
print(wd.upper())
```

```
python
PYTHON
```

In [45]:
```
wd = 'pypthon'
wd.replace('p', 'PP')
```

```
'PPyPPthon'
```

Lists

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets [ ].
- The values stored in a list can be accessed using the slice operator [:]

In [46]:
```
first_lis = ['John', 18, 'M', 'Physics']
print(first_lis)
print(first_lis[ : 2])
print(first_lis[-3 : -1])
```

```
['John', 18, 'M', 'Physics']
['John', 18]
[18, 'M']
```

- Lists type is mutable, i.e., a specific item can be changed
- To change the value of a specific item, refer to the index number

In [47]:
```
my_lis =  ["apple","banana","cherry"]
my_lis[1] = 'grape'
print(my_lis)
```

```
['apple', 'grape', 'cherry']
```

- To determine if a specified item is present in a list using the `in` keyword:

In [48]:
```
fruit_lis =  ['apple','banana','cherry', 'grape']
print('apple' in fruit_lis)
print('orange' in fruit_lis)
```

```
True
False
```

More on String Methods

- `append(obj)`
  - append `obj` to the list
- `insert(pos, obj)`
  - insert `obj` to the location before the specified location `pos`
- `remove(obj)`

- remove the `obj` from the list

check the official tourial (https://docs.python.org/3/tutorial/datastructures.html#more-on-lists) for more details

In [49]:
```
fruit_lis =  ['apple','banana','cherry', 'grape']
fruit_lis.append('orange')
print(fruit_lis)
```

```
['apple', 'banana', 'cherry', 'grape', 'orange']
```

In [50]:
```
fruit_lis =  ['apple','banana','cherry', 'grape']
fruit_lis.insert(1, 'orange')
print(fruit_lis)
```

```
['apple', 'orange', 'banana', 'cherry', 'grape']
```

In [51]:
```
fruit_lis =  ['apple','banana','cherry', 'grape']
fruit_lis.remove('banana')
print(fruit_lis)
```

```
['apple', 'cherry', 'grape']
```

Tuple

- A tuple is another sequence data type that is similar to the list.
- A tuple contains items separated by commas and enclosed within parentheses ().
- Difference between lists and tuples:

- lists are mutable, i.e., specified objects can be modified
- tuples are immutable, objects of tuples cannot be modified

In [52]:
```
fruit_tup = ('apple', 'cherry', 'grape')
print(fruit_tup)
print(fruit_tup[1: 3])
```

```
('apple', 'cherry', 'grape')
('cherry', 'grape')
```

In [53]:
```
'''tupes are immutable, its items cannot be modified
once you try, you will get an error'''

fruit_tup = ('apple', 'cherry', 'grape')
fruit_tup[1] = 'banana'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_628986/493825611.py in <module>
      3
      4 fruit_tup = ('apple', 'cherry', 'grape')
----> 5 fruit_tup[1] = 'banana'

TypeError: 'tuple' object does not support item assignment
```

- To check if a specified item is present in a tupe using the `in` keyword:

In [54]:
```
fruit_lis =  ('apple','banana','cherry', 'grape')
print('apple' in fruit_lis)
print('orange' in fruit_lis)
```

```
True
False
```

Dictionary

- A dictionary is a collection which is unordered, changeable and indexed.
- Consists of a number of key-value pairs.
- It is enclosed by curly braces `{ }`
- objects in dictionary can be also modified, i.e., dictionaries are mutable

In [55]:
```
'''simple example of dict'''

car_dict = {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
print(car_dict)
x = car_dict['model']    # get the value corresponds to the key 'model'
print(x)
car_dict['year'] = 2021
print(car_dict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Mustang
{'brand': 'Ford', 'model': 'Mustang', 'year': 2021}
```

In [56]:
```
'''
we can also add or delete from dictionary
'''


car_dict = {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
car_dict['color'] = 'blue'    # adding new keys and values
print(car_dict)

del car_dict['brand']    # deleting key and values
print(car_dict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'blue'}
{'model': 'Mustang', 'year': 1964, 'color': 'blue'}
```

Sets

- A set is an unordered collection with no duplicate elements
- Set objects also support mathematical operations like union, intersection, difference .....
- Curly braces `{ }` or the `set()` function can be used to create sets
- to create an empty set you have to use `set()` , not `{ }`

In [57]:
```
'''
examples of some operations of sets
'''


fruit = {'apple', 'orange', 'apple', 'pear', 'grape', 'banana'}
print(fruit)
```

```
{'grape', 'banana', 'orange', 'pear', 'apple'}
```

In [58]:
```
characters = set('abcdefaadg')
print(characters)
```

```
{'d', 'e', 'b', 'g', 'a', 'c', 'f'}
```

In [59]:
```
set_A = set('abcdefaadg')
set_B = set('adgjekl')
print(set_A)
print(set_B)

print(set_A & set_B)    # intersection of set_A and set_B
print(set_A | set_B)    # union of set_A and set_B
print(set_A - set_B)    # difference between set_A and set_B
```

```
{'d', 'e', 'b', 'g', 'a', 'c', 'f'}
{'d', 'k', 'e', 'l', 'g', 'j', 'a'}
{'e', 'a', 'd', 'g'}
{'d', 'k', 'e', 'l', 'b', 'g', 'j', 'a', 'c', 'f'}
{'c', 'b', 'f'}
```

User Input From Keyboard

- we can ask Python to pause and read data from the keyboard using the `input()` function
- the `input()` function returns a string
- If we want to read a number from the keyboard, we must convert it from a string to a number using a type conversion function `int()` or `float`

In [60]:
```
'''examples of input from keyboard'''

name = input('Input the name here: ')
print('The name you just input from keyboard is: ', name)
```

```
Input the name here: Susi
The name you just input from keyboard is:  Susi
```

In [61]:
```
'''type convert'''

x = input('please give the stage number in China: ')
y = x - 1
print('the stage number in Europe is: ', y)
```

```
please give the stage number in China: 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_628986/3701886730.py in <module>
      2
      3 x = input('please give the stage number in China: ')
----> 4 y = x - 1
      5 print('the stage number in Europe is: ', y)

TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

In [62]:
```
'''correction'''

x = input('please give the stage number in China: ')
y = int(x) - 1
print('the stage number in Europe is: ', y)
```

```
please give the stage number in China: 5
the stage number in Europe is:  4
```

Assignment Statement

- An assignment statement creates new variables and gives them values.
- An assignment statement consists of an expression on the right-hand side and a variable to store the result.

```
mes = 'Hello, World!'
pi = 3.14
```

Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the	c //= a is equivalent to c = c // a

left operand

Operator Precedence

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence
- Parentheses () have the highest precedence and can be used to force an expression to evaluate in the order you want.
- Exponentiation has the next highest precedence.
- Operators with the same precedence are evaluated from left to right

Readings

- You are required to read Chapter 3 (https://docs.python.org/3/tutorial/controlflow.html) in the Python Officla Tourial
- You are highly encouraged to do some basic exercises from the Internet
  - Here are some examples
    - https://www.w3schools.com/python/python_datatypes.asp (https://www.w3schools.com/python/python_datatypes.asp)
    - https://www.w3schools.com/python/python_datatypes.asp (https://www.w3schools.com/python/python_datatypes.asp)
  - you are also recommended to find your own resources for traning