§1 Data Type

†a Object

Python is an **object-oriented** programing language. Everything is an **object** in Python:

$$\text{object} = \begin{cases} \text{identity}, \\ \text{type / class}, \\ \text{value / state}, \\ \text{methods / behaviors / operations}. \end{cases}$$

```
# print the identity, type, and the value for 4
print(id(4),type(4),4)
# type of any type is a type, the type itself is a type
print(type(type(4)))
print(type(type(type(4))))
```

```
140711773227544 <class 'int'> 4
<class 'type'>
<class 'type'>
```

• **Identity**: it guarantees that different objects have distinct identities at any given time.
• **Type**: objects of the same type support the same operations, and share the same properties.

†b Binding and Input

In Python, the **assignment** of $a = b$ is like making the name a pointing to the object b.

```
# an example for binding
a,b=4,print
print(type(a),a,type(b),b,
      id(a),id(4),id(b),id(print))
b(a+5,"hello")
```

```
<class 'int'> 4 <class 'builtin_function_or_method'>
    <built-in function print> 140723891816984
    140723891816984 2069908885472 2069908885472
9 hello
```

The basic input in Python is through the function input( ). The input takes ONE string as prompt, and it reads input as a string.

```
# an example for input function
n=input(f"{a} and hello\n")
print(type(n),n)
```

```
4 and hello
5
<class 'str'> 5
```

†c Numeric

The following are numeric types:

$$\text{bool} \subset \text{int} \subset_? \text{float} \subset_? \text{complex}$$

```
# an example for the above data types
print(type(True),True,type(1),1,
      type(1.0),1.0,type(1+0j),1+0j)
```

```
<class 'bool'> True <class 'int'> 1 <class 'float'> 1.0
    <class 'complex'> (1+0j)
```

```
# subset example
if True==1==1.0==1+0j:
    print("Yes")
else:
    print("No")
```

```
Yes
```

We can use bool( ), int( ), float( ), and complex( ) to convert a string to the corresponding data type from input( );

```
# input string to number
n=input("type in an integer\n")
print(type(n),n,type(int(n)),int(n))
```

```
type in an integer
17
<class 'str'> 17 <class 'int'> 17
```

identically map from a subset to a larger set, or canonically map from the supset to the restricted set:

```
# identical map and canonical map
print(int(False),float(5),int(3.7))
```

```
0 5.0 3
```

†d More on Bool

```
# logic and bool
print(type(1==0))
print(type(""),bool(""))
if not "":
    print("statement or bool value defined can be used in
        logic")
```

```
<class 'bool'>
<class 'str'> False
statement or bool value defined can be used in logic
```

For statements and numbers, there is a **special method** bool:

```
# special method __bool__()
print(type((5==3).__bool__()),(5==3).__bool__(),
      id((5==3).__bool__()),id(False))
```

```
<class 'bool'> False 140723890821168 140723890821168
```

```
# special method __bool__() for numbers
print(type((0+3.5j).__bool__()),(0+3.5j).__bool__(),
      id((0+3.5j).__bool__()),id(True))
```

```
<class 'bool'> True 140723890821136 140723890821136
```

For some data like string, list etc., the special method len is defined.

```
# sepcial method __len__() for strings
print(type("abcd".__len__()),"abcd".__len__(),
      id("abcd".__len__()),id(4))
```

```
<class 'int'> 4 140723891816984 140723891816984
```

The bool( ) has a **protocol**:

1) If special method bool is defined, then return.
2) Else if the special method is defined, then return True if len is not 0 and vise versa.
3) Else return True.

In general, bool( ) is used for logical determination.

†e More on Float

For float, there are some useful **regular methods**:

```
# regular method is_integer() for floats
print(type((1.3).is_integer()),(1.3).is_integer())
```

```
<class 'bool'> False
```

```
# regular method as_integer_ratio() for floats
print(type((0.5).as_integer_ratio()),(0.5).as_integer_ratio())
```

```
<class 'tuple'> (1, 2)
```