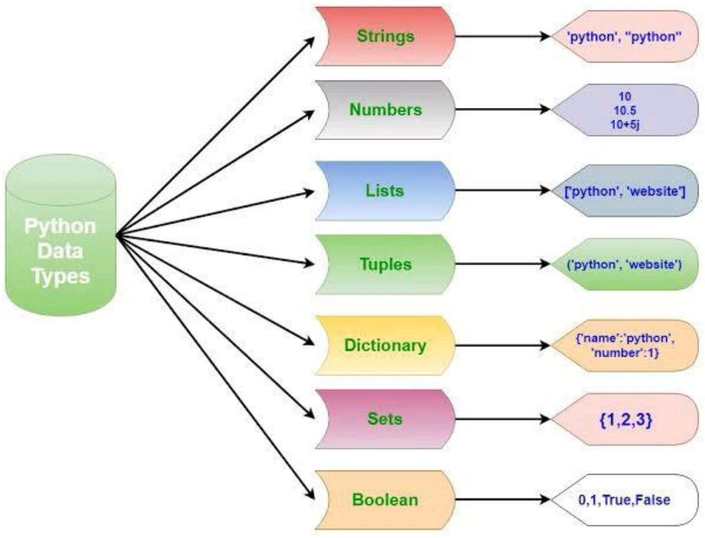
Introduction to Programming in Python

Data Types and Input

Dr. Bill Young
Department of Computer Science
University of Texas at Austin

Last updated: June 4, 2021 at 11:04

Common Python Data Types



What is a Data Type?

A **data type** is a kind of value.

Type	Description	Syntax example
int	An immutable fixed precision number of unlimited magnitude	42
float	An immutable floating point number (system-defined precision)	3.1415927
str	An immutable sequence of characters.	’Wikipedia’ ”Wikipedia” ”””Spanning multiple lines”””
bool	An immutable truth value	True, False
tuple	Immutable, can contain mixed types	(4.0, ’string’, True)
bytes	An immutable sequence of bytes	b’Some ASCII’ b”Some ASCII”
list	Mutable, can contain mixed types	[4.0, ’string’, True, 4.0]
set	Mutable, unordered, no duplicates	{4.0, ’string’, True}
dict	A mutable group of key and value pairs	{’key1’: 1.0, 3: False}

Three Common Data Types

Three data types you’ll encounter in many Python programs are:

`int`: signed integers (whole numbers)

- Computations are exact and *of unlimited size*
- Examples: 4, -17, 0

`float`: signed real numbers (numbers with decimal points)

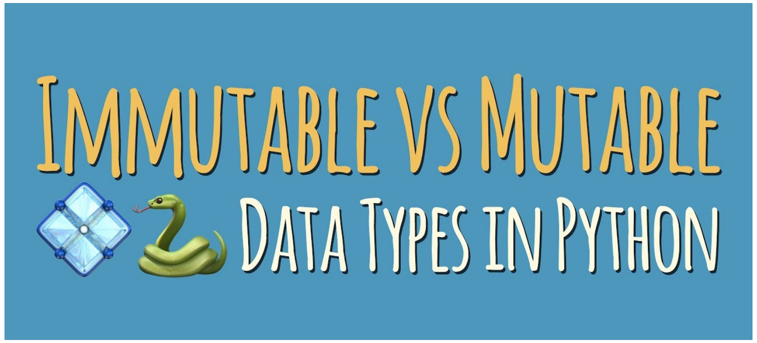
- Large range, but fixed precision
- Computations are approximate, not exact
- Examples: 3.2, -9.0, 3.5e7

`str`: represents text (a string)

- We use it for input and output
- We’ll see more uses later
- Examples: ”Hello, World!”, ’abc’

These are all *immutable.*

Mutable vs. Immutable



An **immutable** object is one that cannot be changed by the programmer after you create it; e.g., numbers, strings, etc.

A **mutable** object is one that can be changed; e.g., sets, lists, etc.

How is Data Stored?

The memory can be thought of as a big array of **bytes**, where a byte is a sequence of 8 bits. Each memory address has an **address** (0..maximum address) and **contents** (8 bits).

...	...	
...	...	
10000	01001010	Encoding for character 'J'
10001	01100001	Encoding for character 'a'
10002	01110110	Encoding for character 'v'
10003	01100001	Encoding for character 'a'
...	...	
...	...	

A byte is the smallest unit of storage a programmer can address. We say that the memory is *byte-addressable*.

Representation Example: ASCII

The standard way to represent *characters* in memory is ASCII. The following is part of the ASCII (American Standard Code for Information Interchange) representation:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	∧	_
96	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	—	}		

The standard ASCII table defines 128 character codes (from 0 to 127), of which, the first 32 are control codes (non-printable), and the remaining 96 character codes are printing characters.

How is Data Stored

Characters or small numbers can be stored in one byte. If data can't be stored in a single byte (e.g., a large number), it must be split across a number of adjacent bytes in memory.

The way data is encoded in bytes varies depending on:

- the data type
- the specifics of the computer

Most of the time, we won't need to know how data is stored in the memory. The computer will take care of that for you.

Data Type Conversion

Warning: the string "25" is *not the same* as the number 25. You can't do arithmetic on strings.

Python provides functions to *explicitly* convert data items from one type to another:

float (< number, variable, string >)
int (<number, variable, string >)
str (<number, variable >)
eval (<string >)

Try not to use eval; it is considered dangerous.

Note: int *truncates*, meaning it throws away the decimal point and anything that comes after it. If you need to *round* to the nearest whole number, use:

round (<number or variable >)

Conversion Examples

```
float(17)
17.0
>>> str(17)
'17'
>>> int(17.75)                    # truncates
17
>>> str(17.75)
'17.75'
>>> int("17")
17
>>> float("17")
17.0
>>> round(17.1)
17
>>> round(17.6)
18
>>> eval("4.3 + 2.5")
6.8
>>> eval(4.3 + 2.5)
TypeError: eval() arg 1 must be a string
```

Keyboard Input

The input() function is used to read data from the user during program execution.

General form:

input (<prompt string >)

When it's called:

- It prints the "prompt string" to the terminal. This is the message to tell the user to enter some input.
- It waits until the user types something and hits "Enter" or "Return."
- It reads in what the user typed *as a string*.

Input Example

```
>>> input("Enter a number: ")
Enter a number: 32
'32'
>>> numEntered = input("Enter a number: ")
Enter a number: 32
>>> numEntered + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> int(numEntered) + 1
33
```

Notice that the error happened because we tried to add a str to an integer.

Remember that keyboard input is always read as a str. Interpret that string as an integer by using the int function.

Exercise: Grade Report with input

Suppose we want to redo our Student Grade Example. Remember we computed a grade report for a specific student, Susie Q. We'd like to write it so that it works for any student. Add `input` statements so you can enter the name and grades.

Try to keep as much of the previous code as possible.

Exercise: Grade Report with input

In file `Grade2.py`:

```
# Constants defining possible points for each exam
# and project:
EXAM1POINTS, EXAM2POINTS, EXAM3POINTS =  100, 90, 65
PROJ1POINTS, PROJ2POINTS = 100, 200

# Enter the student's name:
student = input("Enter student name: ")

# Ask user to input three exam grades:
exam1Grade = int( input("Enter Exam1 grade: " ))
exam2Grade = int( input("Enter Exam2 grade: " ))
exam3Grade = int( input("Enter Exam3 grade: " ))

# Normalize each exam score:
exam1Norm = (exam1Grade / EXAM1POINTS) * 100.0
exam2Norm = (exam2Grade / EXAM2POINTS) * 100.0
exam3Norm = (exam3Grade / EXAM3POINTS) * 100.0

# Compute the average of the three exams:
examAvg = (exam1Norm + exam2Norm + exam3Norm) / 3
```

Exercise: Grade Report with input

```
# Ask user to input two project grades:
proj1Grade = int( input("Enter Proj1 grade: " ))
proj2Grade = int( input("Enter Proj2 grade: " ))

# Normalize each project score:
proj1Norm = (proj1Grade / PROJ1POINTS) * 100.0
proj2Norm = (proj2Grade / PROJ2POINTS) * 100.0

# Compute the average of the two projects:
projAvg = (proj1Norm + proj2Norm) / 2

# COURSE AVERAGE:

# Find the weighted average:
courseAvg = examAvg * 0.6 + projAvg * 0.4
```

Exercise: Grade Report with input

Notice that this code is identical to what we had before.

```
# Print the student's grade report:
print()
print("Grades for", student)
print("   Exam1:", round(exam1Norm, 2))
print("   Exam2:", round(exam2Norm, 2))
print("   Exam3:", round(exam3Norm, 2))
print("Exam average:", round(examAvg, 2))

print("   Proj1:", round(proj1Norm, 2))
print("   Proj2:", round(proj2Norm, 2))
print("Proj average:", round(projAvg, 2))

print("Course average:", round(courseAvg, 2))
```

Running the Program

```
> python Grade2.py
Enter student name: Susie Q.
Enter Exam1 grade: 75
Enter Exam2 grade: 85
Enter Exam3 grade: 57
Enter Proj1 grade: 95
Enter Proj2 grade: 150

Grades for Susie Q.
  Exam1: 75.0
  Exam2: 94.44
  Exam3: 87.69
Exam average: 85.71
  Proj1: 95.0
  Proj2: 75.0
Proj average: 85.0
Course average: 85.43
```