CS303E: Elements of Computers and Programming

Simple Python

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

Last updated: January 17, 2025 at 14:12

Assignment Statements

Assignments are the most common statements in Python programs.

An assignment in Python has form:

Name = Value

This means that variable `name` is *assigned* `value`. I.e., after the assignment, `name` "contains" `value`.

```
>>> x = 17.2
>>> y = -39
>>> z = x * y - 2
>>> print( z )
-672.8
```


Variables

A **variable** is a named memory location used to store values. We'll explain shortly how to name variables.

Unlike many programming languages, Python variables do not have associated types.

```
// C code
int x = 17;     // variable x has type int
x = 5.3;        // illegal
```

```
# Python code
x = 17          # x gets int value 17
x = 5.3         # x gets float value 5.3
```

A variable in Python actually holds a *pointer* to a class object, rather than the object itself.

Types in Python

Is it correct to say that there are no types in Python?

No. It is best to say that Python is "dynamically typed." Variables in Python are untyped, but values have associated types (actually classes).

In some cases, you can convert values of one type to "equivalent" values in another.

Most programming languages assign types to both variables and values. This has its advantages and disadvantages.

Can you guess what the advantages are?

Variables and Assignments

You don't have to *declare* variables, as in many other programming languages. You can create a new variable in Python by assigning it a value.

```
>>> x = 3           # creates x, assigns int
>>> print(x)
3
>>> x = "abc"       # re-assigns x a string
>>> x               # don't really need print
abc                 #   in interactive mode
>>> x = 3.14        # re-assigns x a float
>>> x
3.14
>>> y = 6           # creates y, assigns int
>>> x * y           # uses x and y
18.84
```


Meaning of a Variable

```
x = 17              # Defines and initializes x
y = x + 3           # Defines y and initializes y
z = w               # Runtime error if w undefined
```

This code defines three variables x, y and z. Notice that on the *left hand side* of an assignment the variable is created (if it doesn't already exist), and given a value. On the lhs, it stands for a *memory location.*

On the *right hand side* of an assignment, it stands for the current *value* of the variable. If there is none, it's an error.

Naming Variables

Variable Naming Conventions

Below are (some of) the rules for naming variables:

- Variable names must begin with a letter or underscore ("_") character.
- After that, use any number of letters, underscores, or digits.
- Case matters: "score" is a different variable than "Score."
- You can't use *reserved words*; these have a special meaning to Python and cannot be variable names.

Naming Variables

Python Reserved Words:

and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, nonlocal, None, not, or, pass, raise, return, True, try, while, with, yield

IDLE and many IDEs display reserved words in color to help you recognize them.

Not Reserved, but Don't Use

Function names like `print` are *not* reserved words. But using them as variable names is *a very bad idea* because it redefines them.

```
>>> x = 17
>>> print(x)
17
>>> print = 23
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```

Sometimes you'll use a name, like `str`, not realizing it's a function name. Such errors are sometimes hard to find.

Naming Variables

```
>>> ___ = 10                        # wierd but legal
>>> _123 = 11                       # also wierd
>>> ab_cd = 12                      # perfectly OK
>>> ab|c = 13                       # illegal character
  File "<stdin>", line 1
SyntaxError: can't assign to operator
>>> assert = 14                     # assert is reserved
  File "<stdin>", line 1
    assert = 14
           ^
SyntaxError: invalid syntax
>>> maxValue = 100                  # good one
>>> print = 8                       # legal but ill-advised
>>> print( "abc" )                  # we've redefined print
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```


Naming Variables

In addition to the rules, there are also some conventions that good programmers follow:

- Variable names should begin with a lowercase letter.
- Choose meaningful names that describe how the variable is used. This helps with program readibility.
  - Use `maxValue` rather than `m`.
  - Use `numberOfColumns` rather than `c`.
- One exception is that loop variables are often i, j, etc.

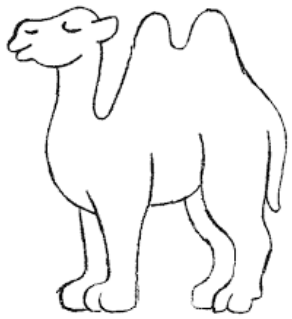
```
    for x in lst: print( x )
```

rather than:

```
    for listItem in lst: print( listItem )
```


Camel Casing

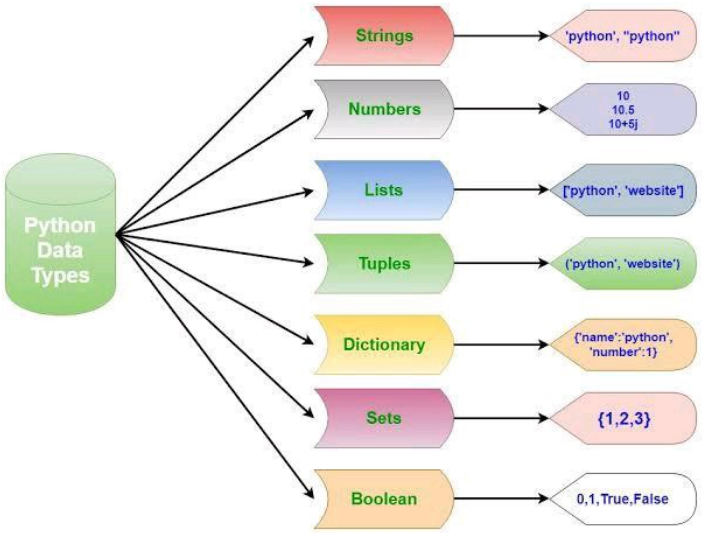
If you use a multi-word names (good practice), a common style is to use "camel casing": `avgHeight`, `countOfItems`, etc.



These are just conventions; many folks use different conventions, and you'll see lots of counterexamples in real code.

Common Python Data Types

Variables don't have types; values do!



What is a Data Type?

A **data type** is a collection of values.

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)

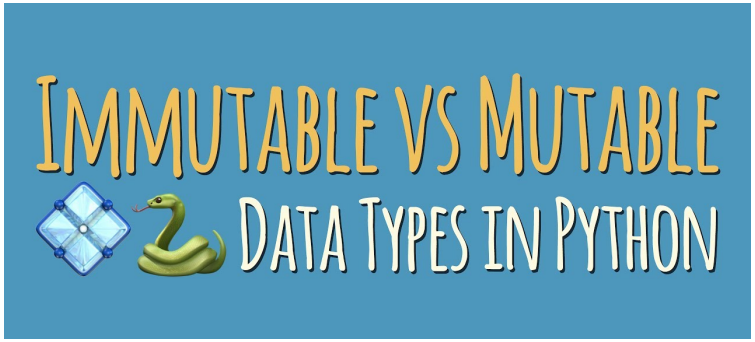
- Large range, but fixed precision
- Computations are *approximate, not exact*
- Examples: 3.2, -9.0, 3.5e7

`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.

What Immutable Means

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

It also means that *there is only one copy of the object in memory.* Whenever the system encounters a new reference to 17, say, it creates a pointer to the already stored value 17.

```
>>> x = 17
>>> y = 12 + 5
>>> z = x
>>> x is y                  # are x and y the "same" value?
True
>>> z is x                  # are x and z the "same" value?
True
```

Variables x, y, z all contain pointers to the same value in memory.

What Immutable Means

If you do something to the object that yields a new value (e.g., uppercase a string), you're actually creating a new object, not changing the existing one.

```
>>> s1 = "abc"              # set s1 to string "abc"
>>> s2 = "ab" + "c"         # set s2 to string "abc"
>>> s3 = s1.upper()         # set s3 to the uppercase of s1
>>> s3
'ABC'
>>> s1 is s2                # s1 and s2 are the same value
True
>>> s1 is s3                # a3 is a different value
False
>>>
```

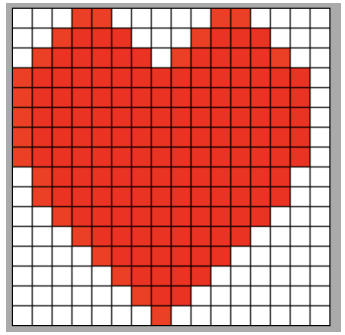

Let's Take a Break

How is Data Stored?

Fundamental fact: *all data* in the computer is stored as a series of bits (0s and 1s) in the memory.

That's true whether you're storing numbers, text, documents, pictures, movies, sounds, programs, etc. *Everything!*

A key problem in designing any computing system or application is deciding how to *represent* the data you care about as a sequence of bits.



Storage Example: Digital Images

For example, images can be stored digitally in any of the following formats (among others):

- JPEG: Joint Photographic Experts Group
- PNG: Portable Network Graphics
- GIF: Graphics Interchange Format
- TIFF: Tagged Image File
- PDF: Portable Document Format
- EPS: Encapsulated Postscript

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.

How is Data Stored?

The memory can be thought of as a big array of **bytes** (1 byte = 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 (American Standard Code for Information Interchange). The following is part of the ASCII 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. **You don't need to know these!**

Some Facts about ASCII

Sorting strings really means sorting according to ASCII code (lexicographically). Here are some things to know:

- upper case letters have consecutive ASCII codes (65 to 90);
- lower case letters have consecutive ASCII codes (97 to 122);
- since upper case letters have smaller codes than lower case, come earlier in a sorted list;
- blank has a low ASCII code so "abc " comes before "abcd" in a sorted list of strings;
- other characters fall at various places in the ASCII table, so sorting strings with non-letter characters is not particularly intuitive.

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.

Formats of Data Types

Notice that the character string "25" *is not* the same as the integer 25.

The integer 25 is represented in binary in the computer by: 00011001. Can you see why?

And the string "25" (two characters) is represented by: 00110010 00110101. Why is that?

Decimal fractions (`float` numbers) are represented in an even more complicated way, since you have to account for an exponent. (Think "scientific notation.") So the number 25.0 (or $2.5 * 10^1$) is represented in yet a third way.

Data Type Conversion

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

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

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
round(17.5)                       # round to even
18
>>> round(18.5)                   # round to even
18
```

Why does round to even make sense?

Arithmetic Operations

Here are some useful operations you can perform on numeric data types.

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float division	1 / 2	0.5
//	Integer division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

(x % y) is often referred to as "x mod y".

Note that "integer division" doesn't always return an integer. E.g., 9 // 2.0 returns 4.0. Expressions that mix an int and float typically return a float.

Simple Program

In file sumDigits.py:

```
""" A simple program that takes a 3 digit integer number in
    variable num, and adds its digits together. """

num = 479
d0 = num % 10              # extract the ones digit
r0 = num // 10             # what's left?

d1 = r0 % 10               # extract the tens digit
r1 = r0 // 10              # what's left?

d2 = r1 % 10               # extract the hundreds digit
r2 = r1 // 10              # quotient should be 0 here

sum = d0 + d1 + d2
print("Sum of the digits of", num, "is", sum)
print("The final quotient is", r2 )
```

Let's run it:

```
> python sumDigits.py
Sum of the digits of 479 is 20
The final quotient is 0
>
```


Augmented Assignment Statements

Python (like C) provides a shorthand syntax for some common assignments:

```
i += j     means the same as     i = i + j
i -= j     means the same as     i = i - j
i *= j     means the same as     i = i * j
i /= j     means the same as     i = i / j
i //= j    means the same as     i = i // j
i %= j     means the same as     i = i % j
i **= j    means the same as     i = i ** j
```

```
>>> x = 2.4
>>> x *= 3.7          # same as x = x * 3.7
>>> print(x)
8.88
```


Mixed-Type Expressions

Most arithmetic operations behave as you would expect for numeric data types.

- Combining two floats results in a float.
- Combining two ints results in an int (except for /). Use // for integer division.
- Dividing two ints gives a float. E.g., 4 / 2 yields 2.0.
- Combining a float with an int usually yields a float.

Python will figure out what the result should be and return a value of the appropriate data type.

Mixed Type Expressions

```
>>> 5 * 3 - 4 * 6       # (5 * 3) - (4 * 6)
-9
>>> 4.2 * 3 - 1.2
11.400000000000002      # approximate result
>>> 5 // 2 + 4          # integer division
6
>>> 5 / 2 + 4           # float division
6.5
```


Special Assignment Statements

Simultaneous assignments:

```
m, n = 2, 3
```

means the same as:

```
m = 2
n = 3
```

With the caveat that these happen *at the same time*.

What does the following do?

```
i, j = j, i
```

Multiple assignments:

```
i = j = k = 1
```

means the same as:

```
k = 1
j = k
i = j
```

Note that these happen right to left.

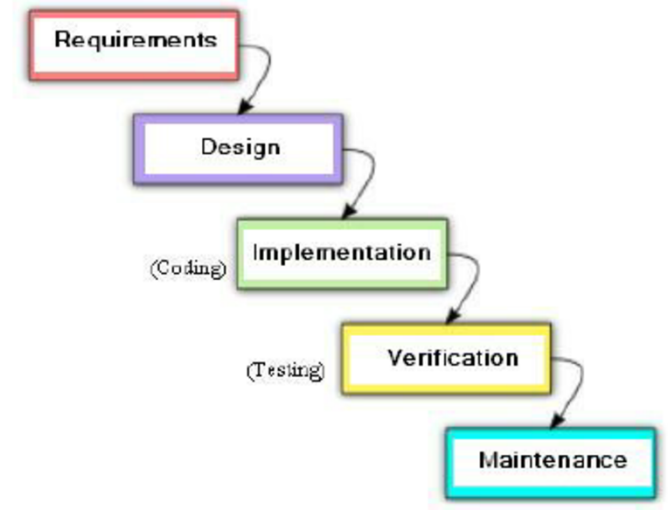
Advice on Programming

Think before you code!
Think before you code!
Think before you code!

- Don't jump right into writing code.
- Think about the overall process of solving your problem; write it down.
- Refine each part into subtasks. Subtasks may require further refinement.
- Code and test each subtask before you proceed.
- Add `print` statements to view intermediate results.

Advice on Programming

The software development process outlined in Section 2.13 is called *the waterfall model*. You'll do well to follow it, except on the simplest programs.



Next stop: More Simple Python.