Python Data Types (I)

- Expressions, Variables, and Assignments
- Integers
- Floats
- Booleans
- Strings

What this course is about

Designing computational models

A computational model is a simplified description of some piece of the world, using exactly defined data types and actions

Thinking algorithmically

An algorithm is a precise, step-by-step solution to a problem

Writing programs

A program is written in a computer language (e.g., Python) and implements an algorithm

Our tools

A computer language (Python)

Instant task: Go to python.org and read about the language, see how to download Python

The Python shell

Instant task: start the Python shell and type in a few expressions

The Python editor

Instant task: open a new window, write some text and save the file

Computer language v. natural language

Natural language (e.g., English, Arabic, Chinese)

- Very many words
- Evolved, not designed. Meaning is derived by association and usage.
- Has grammatical structure, but very permissive of mistakes, variation and ambiguity
- Meant for a human being

Programming (computer) language

- Very few words (a few dozen in Python)
- Meaning is exact and is assigned by the language designer
- Very persnickety about accuracy. A single mistake destroys the correctness of a program.
- Can be executed by a special type of machine (a computer)

Arithmetic data types

Counting type (int)

- A type of thing that is discrete and can be counted
- Similar to the integer in mathematics
- The value is made up of the digits 0-9 and (optionally) a sign

```
>>> numberOfLittlePigs = 3
>>> numberOfLittlePigs
3
>>> type(numberOfLittlePigs)
class <'int'>
```

Floating-point arithmetic type

- Can have a non-unit value
- Similar to (but not exactly the same as) a decimal number in mathematics
- The value is made up of the digits 0-9, a decimal point and (optionally) a sign

```
>>> pi = 3.14159
>>> type(pi)
class <'float'>
```

Arithmetic expressions

The Python interactive shell can be used to evaluate arithmetic expressions

`14//3` is the quotient when 14 is divided by 3 and `14%3` is the remainder. Note: // is integer division.

`2**3` is 2 to the power 3

`abs()`, `min()`, and `max()` are functions

- `abs()` takes a number as input and returns its absolute (unsigned) value
- `min()` (resp., `max()`) takes an arbitrary number of inputs and return the "smallest" ("largest") among them

```
>>> 2 + 3
5
>>> 7 - 5
2
>>> 2*(3+1)
8
>>> 5/2
2.5
>>> 5//2
2
>>> 14//3
4
>>> 14%3
2
>>> 2**3
8
>>> abs(-3.2)
3.2
>>> min(23,41,15,24)
15
>>> max(23,41,15,24)
41
```

Be Careful using == with Floats

Floats are very accurate approximations to real numbers, however:

When you are dealing with multiple arithmetic operations on fractional numbers, infinitesimally small errors get propagated.

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.5
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.6
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.7
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.7999999999999999
>>> 0.8 == 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
False
>>>
```

We will revisit this issue when we discuss conditional statements.

Boolean (logical, T/F) data type

In addition to algebraic expressions, Python can evaluate Boolean (T/F) expressions

- A Boolean expression evaluates to `True` or `False`
- Boolean expressions often involve comparison operators
  <, >, ==, !=, <=, and >=

```
>>> 2 < 3
True
>>> 2 > 3
False
>>> 2 == 3
False
>>> 2 != 3
True
>>> 2 <= 3
True
>>> 2 >= 3
False
>>> 2+4 == 2*(9/3)
True
```

In an expression containing arithmetic and comparison operators:

- Arithmetic operators are evaluated first
- Comparison operators are evaluated next

Boolean operators

Python can evaluate more complex Boolean expressions

- Every Boolean expression evaluates to `True` or `False`
- Boolean expressions may be combined using Boolean operators `and`, `or`, and `not`

In an expression containing algebraic, comparison, and Boolean operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next
- Boolean operators are evaluated last

```
>>> 2<3 and 3<4
True
>>> 4==5 and 3<4
False
>>> False and True
False
>>> True and True
True
>>> 4==5 or 3<4
True
>>> False or True
True
>>> False or False
False
>>> not(3<4)
False
>>> not(True)
False
>>> not(False)
True
>>> 4+1==5 or 4-1<4
True
```

Exercise

Translate the following into Python algebraic or Boolean expressions and then evaluate them:

a) The difference between Annie's age (25) and Ellie's (21)
b) The total of $14.99, $27.95, and $19.83
c) The area of a 20 x 15 rectangle
d) 2 to the 10th power
e) The minimum of 3, 1, 8, -2, 5, -3, and 0
f) 3 equals 4-2
g) The value of 17//5 is 3
h) The value of 17%5 is 3
i) 284 is even
j) 284 is even and 284 is divisible by 3
k) 284 is even or 284 is divisible by 3

```
>>> 25 - 21
4
>>> 14.99 + 27.95 + 19.83
62.769999999999996
>>> 20*15
300
>>> 2**10
1024
>>> min(3,1,8,-2,5,-3,0)
-3
>>> 3 == 4-2
False
>>> 17//5 == 3
True
>>> 17%5 == 3
False
>>> 284%2 == 0
True
>>> 284%2 == 0 and 284%3 == 0
False
>>> 284%2 == 0 or 284%3 == 0
True
```

Variables and assignments

Just as in algebra, a value can be assigned to a variable (name), such as *x*

When variable *x* appears inside an expression, it evaluates to its assigned value

A variable (name) does not exist until it is assigned

The assignment statement has the format

```
<variable> = <expression>
```

<expression> is evaluated first, and the resulting value is assigned to variable <variable>

```
>>> x = 3
>>> x
3
>>> 4*x
12
>>> y
Traceback (most recent call
last):
  File "<pyshell#59>", line
1, in <module>
    y
NameError: name 'y' is not
defined
>>> y = 4*x
>>> y
12
```

Naming rules

(Variable) names can contain these characters:

- a through z
- A through Z
- the underscore character _
- digits 0 through 9

Names cannot start with a digit. Names can start with a capital letter or underscore, but don't do it.

For a multiple-word name, use

- either the underscore as the delimiter
- or *camelCase* capitalization

Meaningful names are important. They make your program more readable by a human being (such as you).

```
>>> my_x2 = 21
>>> my_x2
21
>>> 2x = 22
SyntaxError: invalid
syntax
>>> new_temp = 23
>>> newTemp = 23
>>> counter = 0
>>> temp = 1
>>> price = 2
>>> age = 3
```

Text data type (string)

In addition to number and Boolean values, Python support string values

A string value is represented as a sequence of characters enclosed within single or double quotes

A string value can be assigned to a variable

String values can be manipulated using string operators and functions

```
>>> 'Hello, World!'
'Hello, World!'
>>> s = 'rock'
>>> t = 'climbing'
>>> s + ' ' + t
>>> rock climbing
```

String operators

Usage	Explanation
`x in s`	`x` is a substring of `s`
`x not in s`	`x` is not a substring of `s`
`s + t`	Concatenation of `s` and `t`
`s*n, n*s`	Concatenation of `n` copies of `s`
`s[i]`	Character at index `i` of `s`
`len(s)`	(function) Length of string `s`

To view all operators, use the `help()` tool

```
>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(string[, encoding[, errors]]) -> str
...
```

```
>>> s = 'rock'
>>> t = 'climbing'
>>> s == 'rock'
True
>>> s != t
True
>>> s < t #explain this answer!
False
>>> s > t
True
>>> s + t
'rockclimbing'
>>> s + ' ' + t
'rock climbing'
>>> 5 * s
'rockrockrockrockrock'
>>> 20 * '_'
'____________________'
>>> 'o' in s
True
>>> 'o' in t
False
>>> 'bi' in t
True
>>> len(t)
8
```

Exercise

Write Python expressions that create strings `s1 ('good')`, `s2 ('bad')`, and `s3 ('silly')` that correspond to:

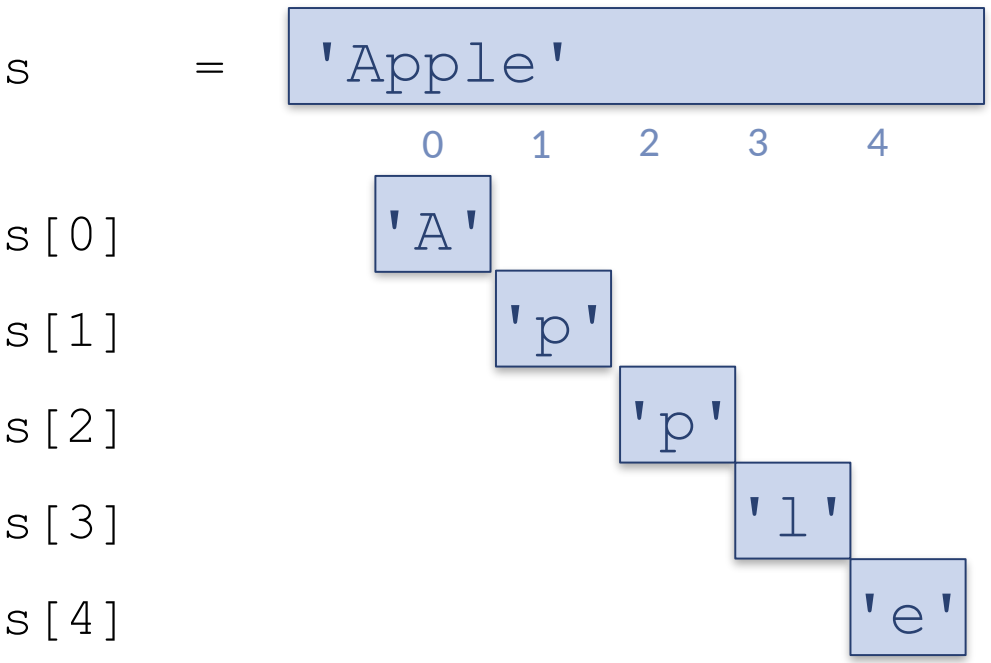
a) `'ll'` appears in `s3`
b) the blank space does not appear in `s1`
c) the concatenation of `s1`, `s2`, and `s3`
d) the blank space appears in the concatenation of `s1`, `s2`, and `s3`
e) the concatenation of 10 copies of `s3`
f) the total number of characters in the concatenation of `s1`, `s2`, and `s3`

```
>>> s1
'good'
>>> s2
'bad'
>>> s3
'silly'
>>> 'll' in s3
True
>>> ' ' not in s1
True
>>> s1 + s2 + s3
'goodbadsilly'
>>> ' ' in s1 + s2 + s3
False
>>> 10*s3
'sillysillysillysillysillysillysillysillysillysillysillysillysillysillysilly'
>>> len(s1+s2+s3)
12
>>>
```

Index and indexing operator

The index of an item in a sequence is its position with respect to the first item

- The first item has index 0
- The second has index 2
- The third has index 2 …



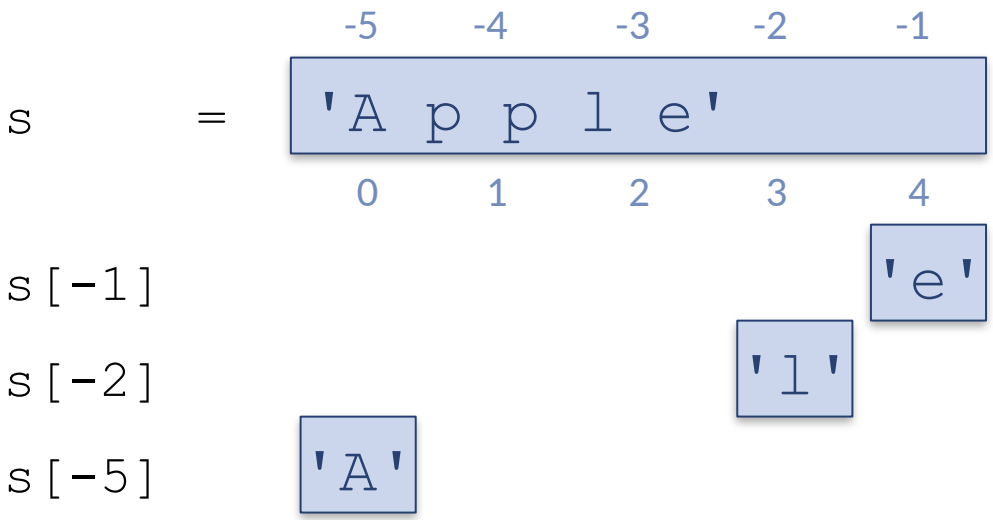
The indexing operator `[]` takes a index `i` and returns a substring consisting of the single character at index `i`

```
>>> s = 'Apple'
>>> s[0]
'A'
>>> s[1]
'p'
>>> s[4]
'e'
```

Negative index

A negative index is used to specify a position with respect to the "end"

- The last item has index -1
- The second to last item has index -2
- The third to last item has index -3 …




```
>>> s = 'Apple'
>>> s[-1]
'e'
>>> s[-2]
'l'
>>> s[-5]
'A'
```

Exercise

String `s` is defined to be

```
'abcdefgh'
```

Write expressions using `s` and the indexing operator `[]` that return the following strings:

a) `'a'`
b) `'c'`
c) `'h'`
d) `'f'`

```
>>> s = 'abcdefgh'
>>> s[0]
'a'
>>> s[2]
'c'
>>> s[7]
'h'
>>> s[-1]
'h'
>>> s[-3]
'f'
>>>
```