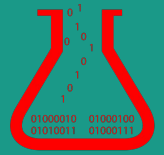
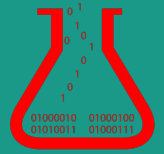
Variable Types

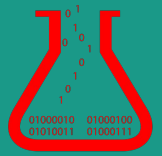


Python Primitive Types

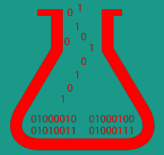


Agenda

Intro & Basics
Integers
Floats
Working with Int and Float
Booleans
Strings
Working with Strings

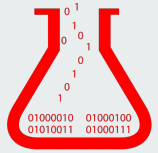


Intro & Basics



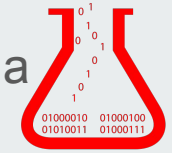
Variables

- Python uses variables to store information
- Syntax to assign a value to a variable is the equal sign
  - For example: count = 6
  - Assigns the integer value 6 to the variable named count
- Python is dynamically typed:
  - Dynamically typed means that Python doesn't need to know the type of the variable - the variable name just 'points' to that object and the object has a type associated with it



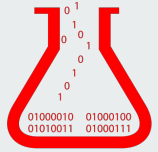
Variables - Dynamic Typing

- Python is dynamically typed:
  - Dynamically typed means that Python doesn't need to know the type of the variable on initialization
  - The variable name just 'points' to that object and the object has a type associated with it
  - For example: count = 6 (count is an integer)
  - And then we do: count = 'six' (count is now a string)
  - Python is ok with this change and won't give an error
- Opposite is strongly typed where you have to define a type for each variable and then that variable name is only associated with that type
  - For example: count: int = 6
  - 'count' is always of type 'integer' and will get an error if assigned to a different type



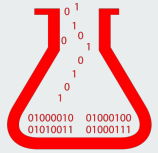
Variable Naming

- Case matters (mystr is a different variable than MyStr)
- Cannot start with a number
- Usually variable names are in all lowercase
- Can use underscores to make them more readable
    - E.g.: word_dict or my_list
- Keep variable names short (you might have to write them a lot!)
- ‘Counter’ variables are often a single letter like: i,j,k
- Try to name variables something that is easy to read for you and other programmers (i.e., avoid lowercase “L” as it looks like uppercase “I” and pipe: l, I, |)



= VS ==

- = is the assignment operator
  - That is = assigns the values on the right to the variable name on the left
- == is the equality operator
  - == sees if the value on the left is the same as the value on right
    - Returns True if the values are equal
    - Returns False if the value are not equal
- Examples:
  - 3 == 3.0 (True)
  - True == 1 (True)
  - 4 == ‘4’ (False)

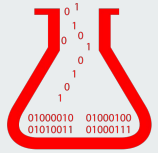


Mutability

Mutable = the item can be changed after created

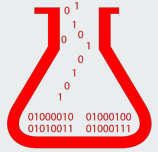
Immutable = the item cannot be changed after created

All primitives are immutable

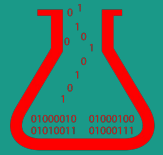


Python's Primitive (Atomic) Types

- Integer (int)
- Float
- Boolean (bool)
- NoneType
- String (str)



Integer



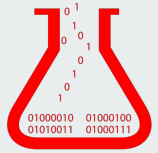
Integer

Positive or negative whole numbers with no decimal point

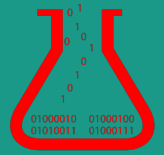
Examples: -1001, 0, 6

Python will dynamically allocated the needed memory for any size of integer (min/max is only limited by the memory size of your machine)

int() creates an empty integer (defaults to 0)



Float



Float

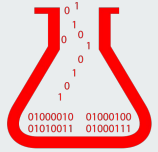
Represent real numbers and are written with a decimal point dividing the integer and fractional parts.

Floats may also be in scientific notation

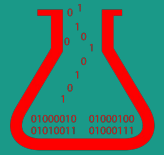
Examples: -2.5, 3.1415, 7.67, 1.4e6, 2.1e10

Again, Python will dynamically allocated the needed memory for any size of float

float() creates an empty float (defaults to 0.0)

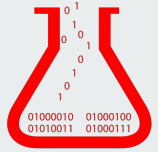


Working with Int and Float



Integer & Float Operators

+ add
- subtract
* multiple
/ divide (if used on two integers will return a float)
// integer divide (gives back divisor) 5 // 2 = 2
% modulo (gives back remainder) 5.0 % 2 = 1.0
** raise to the power 3 ** 2 = 9



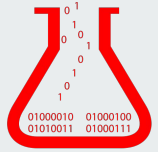
Exercise: Integer & Float Operators

What would 8 // 2 = ?

How would you get the remainder of 12.0 divided by 5?

How would you do the square root of 16?

What is the result of (⅓ + 1000) - 1000 = ?



Exercise: Integer & Float Operators

What would 8 // 2 = ?

4

How would you get the remainder of 12.0 divided by 5?

2.0

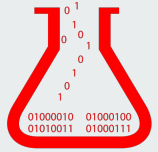
How would you do the square root of 16?

16 ** 0.5

What is the result of (⅓ + 1000) - 1000 = ?

0.3333333333337123 (or something like that)

Why isnt 0.3333333333333333?

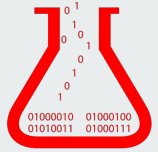


Floating Point Error

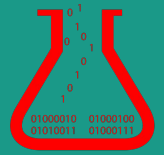
The reason float numbers are not exact is due to Floating Point Error.

A computer can only store numbers as 0/1 so as they get smaller it is more difficult to store the exact number when doing multiple calculations

This is a point to remember when comparing floats!



Boolean



Boolean

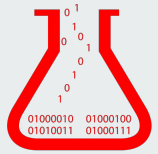
True or False

Equivalent to [1,0] (respectively):

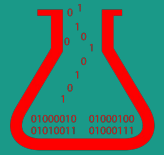
- False evaluates to 0
- True evaluates to 1

Can also do other operations (like adding): 2 + True = 3

bool() creates an empty boolean (defaults to False)

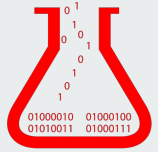


NoneType

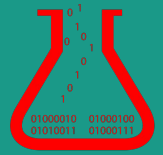


NoneType

None is the Python equivalent of Null or NA
It is **not** equivalent to 0 or False:
None == 0, or None == False, both evaluate to False
Note: None becomes ‘None’ if type case as a string with str()



String



Strings

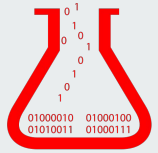
A string in Python is a sequence of characters

Can use single quotes (‘) or double quotes (“)

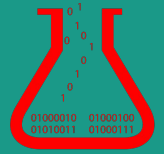
Can use the opposite quotes if the string contains the other type of quote

Examples: “w”, “Dog”, “h7jb67”, ‘four’, ‘4’, “The cat in the hat.” “John’s dog”

Strings are also an iterable data type (a type that can be used in a sequential fashion to find the next item)



Working with Strings



String Operators

str(), " " or ' ' - creates an empty string variable

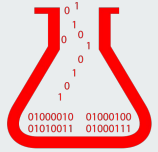
There is no char type, just a string with length 1

Concatenation (adding two strings together):

“mystring” + “anotherstring” → “mystringanotherstring”

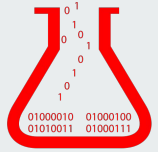
Multiply: “mystring” * 3 → 'mystringmystringmystring'

""" """ - triple quotes, used for block comments on multiple lines
Example: """This is a long string that is
split over two lines """



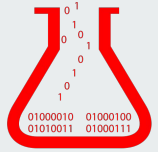
String Methods

- upper(), .lower(), .capitalize(), .title()
  - mystring = ‘Abc dEf’
  - mystring.**upper()** → ‘ABC DEF’
    - Uppercase all letters
  - mystring.**lower()** → ‘abc def’
    - Lowercase all letters
  - mystring.**capitalize()** → ‘Abc def’
    - Capitalizes first letter of first word
  - mystring.**title()** → ‘Abc Def’
    - Capitalizes first letter of each word



Advanced String Methods

- **str.strip()**
  - Used to remove ’unwanted’ characters from the start & end of a string:
    - mystr = "----Our string we want----"
    - mystr.strip(‘-’) ⟶ ‘Our string we want’
- **str.split()**
  - Used to split a string into multiple items on a character - returns a list of those strings
    - mystr = “eat, drink, sleep”
    - mystr.split(‘,’) ⟶ [‘eat’, ‘drink’, ‘sleep’]



Questions?

Contact:

Denis Vrdoljak
denis@bds.group