ME / MSE 241
Engineering Computations

Fall 2022

Instructor: Narasimha Boddeti

narasimha.boddeti@wsu.edu, Sloan 213

Recap

- Writing and executing Python code
  - Interactive
    - Running the Python interpreter from the terminal (or the command prompt on Windows)
    - Jupyter notebook console on a local or remote Jupyter Lab server
  - Scripts
    - Run through the python interpreter
      - From the command line – "*python script.py*"
  - Jupyter Notebooks
    - Local or remote Jupyter Lab server
    - Visual studio code

Python Basics

- Built-in data types in Python
  - Numeric types
  - Boolean type
  - Sequence types
    - For sequences of numerical/textual data
  - …
- Programmers can also create their own data types through *classes*
    - A *class* defines the blueprint/template for creating new *instances* of *objects*
    - *Objects* combine *data* (i.e., attributes of the object) with *methods* (i.e., functions that operate on the object's data)
    - Examples
      - Squares, rectangles and rhombi are *instances* of the *class* quadrilaterals
      - Basketball, volleyball, tennis, hockey etc. are *instances* of the *class* sports

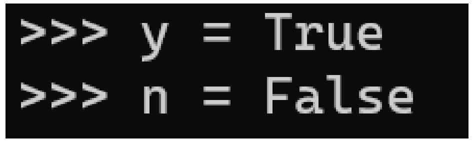
Data Types

- Built-in basic data types in Python:
  - *bool* – Boolean data
  - *int*, *float*, *complex* – numbers
  - *str* – sequences of text termed strings

- In Python, all data is represented by *objects*
  - Each object has an *identity*, *type*, and *value*
  - This contrasts with languages C and C++ where basic data types like *bool, int*, and *float* are just data

Data Types: Boolean & Integer

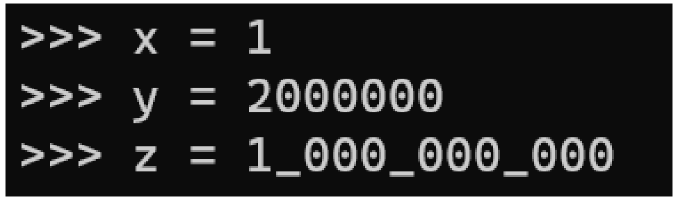
- Boolean type – *bool*
  - Takes a value of either *True* or *False*



```
>>> y = True
>>> n = False
```

Note: **>>>** is the prompt for input in the Python interpreter (not a part of the Python statement)

- Integer type – *int*
  - Integers of arbitrary size
  - Note: the *int* data type in C/C++ is limited to 4 bytes (*long* is 8 bytes)

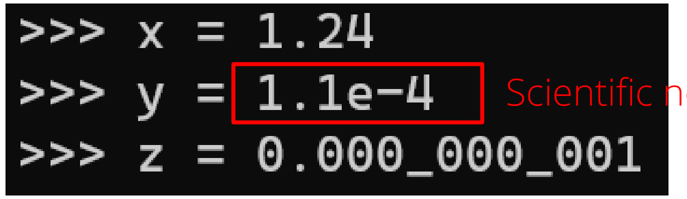


```
>>> x = 1
>>> y = 2000000
>>> z = 1_000_000_000
```

  - "_" can be used to separate digits for clarity

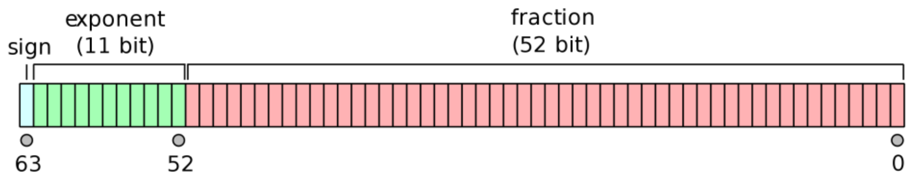
Data Types: Float

- Float type – *float*
  - Real numbers of size 8 bytes (same as the data type *double* in C/C++)



```
>>> x = 1.24
>>> y = 1.1e-4
>>> z = 0.000_000_001
```

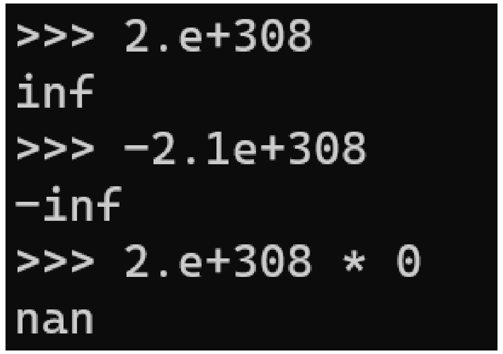
Scientific notation



Source: Wikipedia.org

- Max. float value ≈ 1.7977e+308
- Min. positive float value = 5e-324

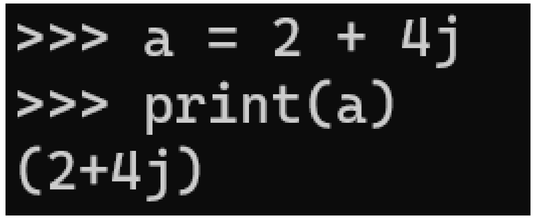
- Floating point numbers *inf* and *nan*
  - *inf* – any number that goes beyond the memory capacity of a *float*
    - Short for infinity
  - *nan* - numbers that lack mathematical basis
    - Short for “not a number”



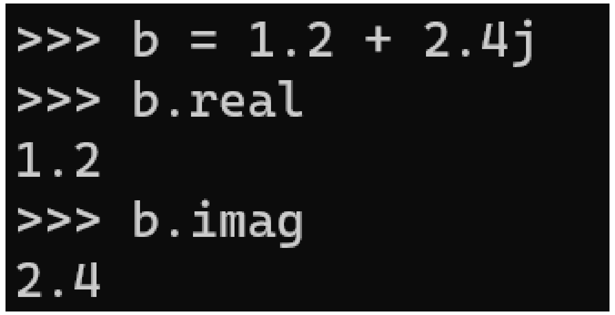
```
>>> 2.e+308
inf
>>> -2.1e+308
-inf
>>> 2.e+308 * 0
nan
```

Data Types: Complex

- Complex type – *complex*
  - Represents complex numbers through two *floats*
  - Written as the sum of the real part and imaginary part
  - The numeric value of the imaginary part should be immediately followed by the letter "j" which denotes the imaginary unit
    - $j^2 = -1$



```
>>> a = 2 + 4j
>>> print(a)
(2+4j)
```



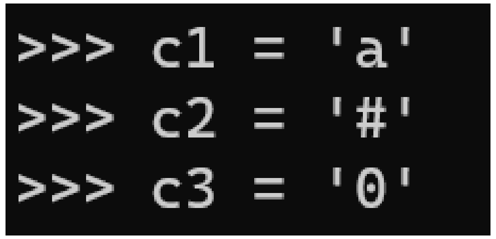
```
>>> b = 1.2 + 2.4j
>>> b.real
1.2
>>> b.imag
2.4
```

`real` and `imag` are attributes specific to the complex data type

Data Types: String

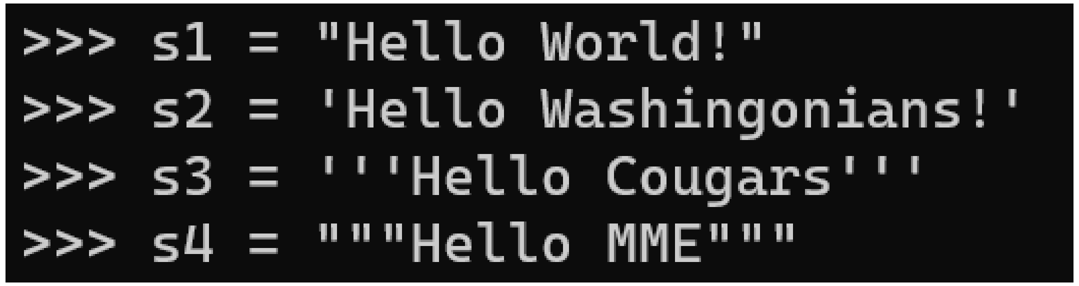
- Text type – *str*
  - Represents a sequence/string of characters
  - Characters include:
    - Alphabet – lowercase and uppercase
    - Digits – 0, 1, 2, ⋯, 9
    - Symbols - !, @, #, $, ⋯
    - Special characters:

Line feed or new line	\n
Form feed or page break	\f
Carriage return	\r
Tab	\t
Backspace	\b
Bell	\a



```
>>> c1 = 'a'
>>> c2 = '#'
>>> c3 = '0'
```

While C/C++ have a *char* data type for single characters, there is no such thing in Python

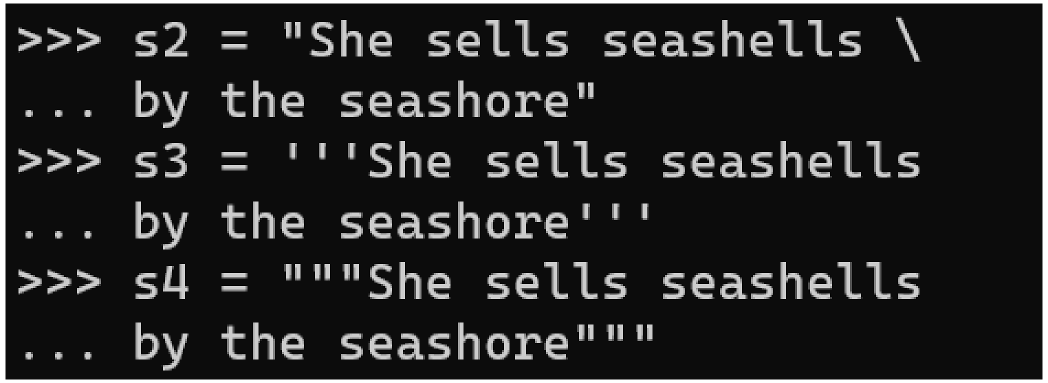


```
>>> s1 = "Hello World!"
>>> s2 = 'Hello Washingonians!'
>>> s3 = '''Hello Cougars'''
>>> s4 = """Hello MME"""
```

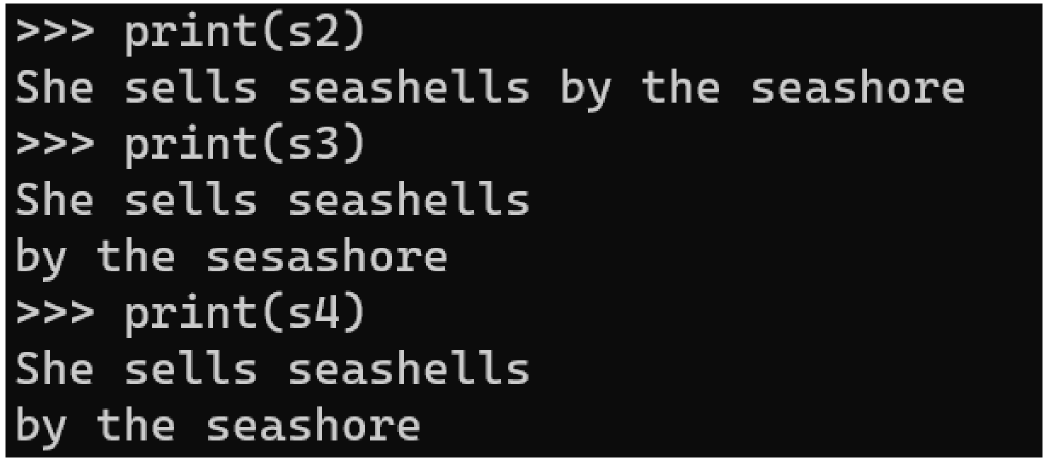
Single, double and triple quotes can be used to define strings but cannot be mixed

Data Types: String

- With triple single/double quotes, a string can be split into multiple lines

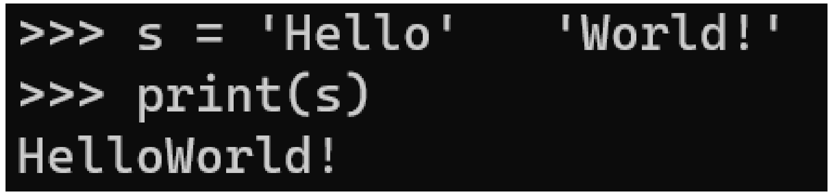


```
>>> s2 = "She sells seashells \
... by the seashore"
>>> s3 = '''She sells seashells
... by the seashore'''
>>> s4 = """She sells seashells
... by the seashore"""
```



```
>>> print(s2)
She sells seashells by the seashore
>>> print(s3)
She sells seashells
by the sesashore
>>> print(s4)
She sells seashells
by the seashore
```

Note:
- The backslash splits the Python statement into two lines and is not part of the string
- Triple single/double quotes allow splitting and include any whitespace and newlines

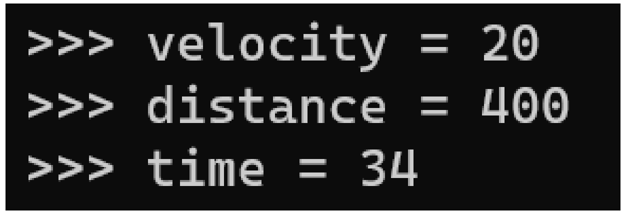


```
>>> s = 'Hello'    'World!'
>>> print(s)
HelloWorld!
```

String definitions can be split by spaces, the Python interpreter will ignore them

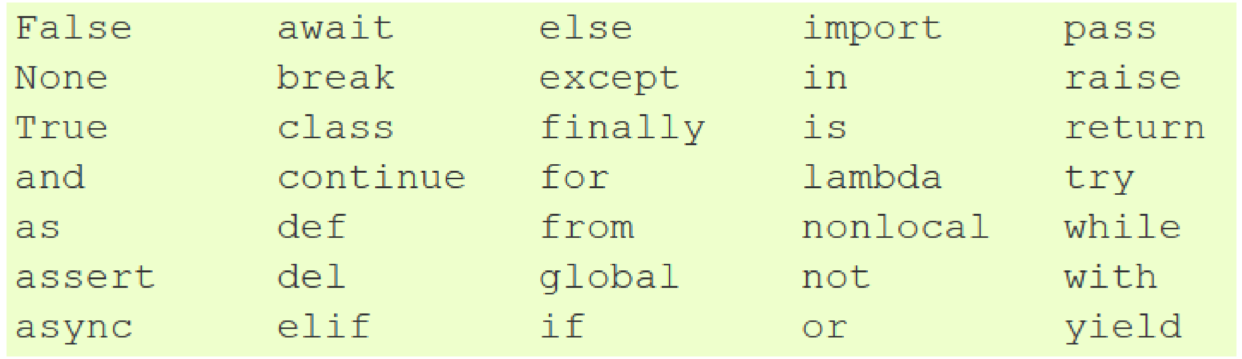
Identifiers

- Variables of different data types are typically assigned a name or *identifier*



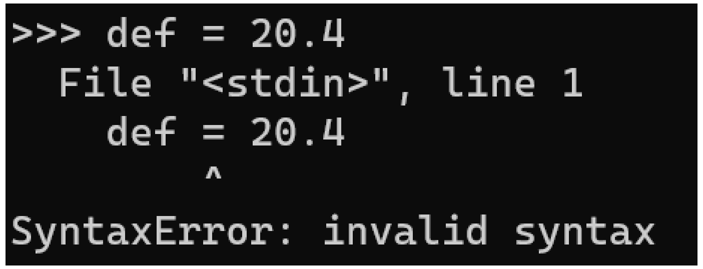
```
>>> velocity = 20
>>> distance = 400
>>> time = 34
```

- Cannot use any of the Python keywords for a variable name



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

Python keywords



```
>>> def = 20.4
  File "<stdin>", line 1
    def = 20.4
        ^
SyntaxError: invalid syntax
```

Interpreter throws an *exception* or error if an attempt is made to use a keyword for a variable name

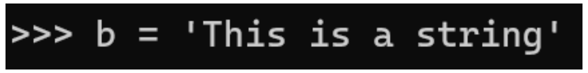
Identifiers

- Rules for identifiers
  - Can be of any length
  - Can combine English alphabet (both cases, a to z and A to Z) and digits (0 to 9) with underscore (_)
    - A_1, b2, b_2
    - _b, __b
    - This_is_an_identifier
    - …
  - Identifier cannot begin with a digit (e.g., 1a)
- Underscore usage conventions
  - A trailing underscore is used with a variable if it conflicts with a keyword
  - A single underscore is a temporary variable
    - Also stores the last evaluated expression when using Python interpreter interactively
  - Identifiers with leading and trailing double underscores and a leading underscore mean something when used in the definition of Python class attributes

Naming Convention

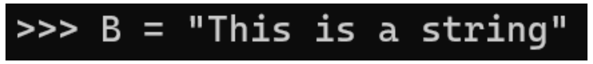
- A consistent naming scheme keeps the source code readable and thus, maintainable
- A document named “PEP 8 – Style Guide for Python Code” describes some commonly used conventions

  - Single lowercase letter



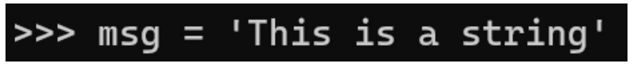
```
>>> b = 'This is a string'
```

  - Single uppercase letter



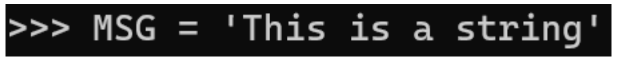
```
>>> B = "This is a string"
```

  - lowercase



```
>>> msg = 'This is a string'
```

  - UPPERCASE

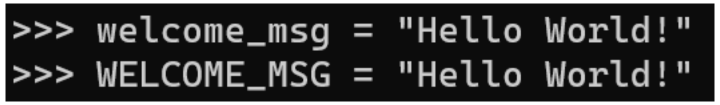


```
>>> MSG = 'This is a string'
```

Naming Convention

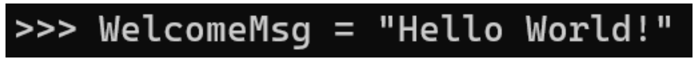
- A consistent naming scheme keeps the source code readable and thus, maintainable
- A document named "PEP 8 – Style Guide for Python Code" describes some commonly used conventions

  - Snake case
    - Words connected by underscore



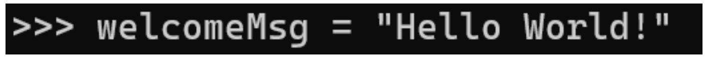
```
>>> welcome_msg = "Hello World!"
>>> WELCOME_MSG = "Hello World!"
```

  - Camel case
    - Starting of each word capitalized



```
>>> WelcomeMsg = "Hello World!"
```

  - Mixed case
    - First word is not capitalized
    - Else, same as camel case



```
>>> welcomeMsg = "Hello World!"
```