First Session
Intro to Python (Numpy, Pandas, Matplotlib)

GitHub Campus Teams - AUC
Machine Learning Committee

AUC, Spring 2022
Abdelrahman Fawzy,
Youssef Hussien

Brief History of Python

- ❏ Invented in the Netherlands, early 90s by Guido van Rossum
- ❏ Named after Monty Python
- ❏ Open sourced from the beginning
- ❏ Considered a scripting language, but is much more
- ❏ Scalable, object oriented and functional from the beginning
- ❏ Used by Google from the beginning
- ❏ Increasingly popular

Official Documents

- For more information about Python everything is documented here:
  - http://docs.python.org/
- A very good tutorial
  - https://docs.python.org/3/tutorial/index.html

Installing Python

- Python is pre-installed on most Unix systems, including Linux and MAC OS X
- The pre-installed version may not be the most recent one (2.6.2 and 3.1.1 as of Sept 09)
- Download from http://python.org/download/
- Python comes with a large library of standard modules
- There are several options for an IDE
  - IDLE – works well with Windows
  - Emacs with python-mode or your favorite text editor
  - Eclipse with Pydev (http://pydev.sourceforge.net/)

Running Python on MAC and UNIX

- Call python program via the python interpreter

```
% python fact.py
```

- Make a python file directly executable by
- Adding the appropriate path to your python interpreter as the first line of your file

```
#!/usr/bin/python
```

- Making the file executable

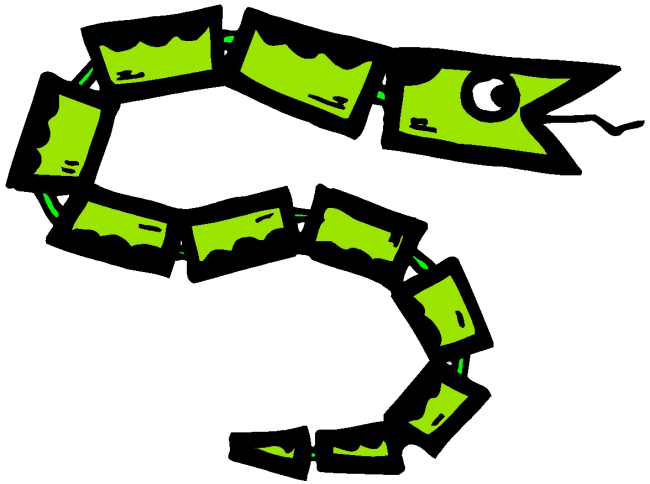
```
% chmod a+x fact.py
```

- Invoking file from Unix command line

```
% fact.py
```

The Basics

```
x = 34 - 23            # A comment.
y = “Hello”            # Another one.
z = 3.45
if z == 3.45 or y == “Hello”:
    x = x + 1
    y = y + “ World”   # String concat.
print x
print y
```



A Code Sample (in IDLE)

```
x = 34 - 23              # A comment.
y = “Hello”              # Another one.
z = 3.45
if z == 3.45 or y == “Hello”:
    x = x + 1
    y = y + “ World”   # String concat.
print x
print y
```

Enough to Understand the Code

- Indentation matters to code meaning
  - Block structure indicated by indentation
- First assignment to a variable creates it
- Variable types don’t need to be declared.
  - Python figures out the variable types on its own.
- Assignment is = and comparison is ==
- For numbers + - * / % are as expected
  - Special use of + for string concatenation and % for string formatting (as in C’s printf)
- Logical operators are words (and, or, not) not symbols
- The basic printing command is print

Basic Data Types

- **Integers (default for numbers)**

  z = 5 / 2 # Answer 2, integer division

- **Floats**

  x = 3.456

- **Strings**
  - Can use “” or ‘’ to specify with “abc” == ‘abc’
  - Unmatched can occur within the string: “matt’s”
  - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them:
    “””“a‘b“c”””

Whitespace

Whitespace is meaningful in Python: especially indentation and placement of newlines

- Use a newline to end a line of code

  Use \ when must go to next line prematurely

- No braces {} to mark blocks of code, use *consistent* indentation instead
  - First line with *less* indentation is outside of the block
  - First line with *more* indentation starts a nested block
- Colons start of a new block in many constructs, e.g. function definitions, then clauses

Comments

- Start comments with #, rest of line is ignored
- Can include a “documentation string” as the first line of a new function or class you define
- Development environments, debugger, and other tools use it: it’s good style to include one

```
def fact(n):
    """fact(n) assumes n is a positive integer and
    returns factorial of n."""
    assert(n>0)
    return 1 if n==1 else n*fact(n-1)
```

Assignment

- *Binding a variable* in Python means setting a *name* to hold a *reference* to some *object*
  - *Assignment creates references, not copies*
- Names in Python do not have an intrinsic type, objects have types
  - Python determines the type of the reference automatically based on what data is assigned to it
- You create a name the first time it appears on the left side of an assignment expression:

  x = 3
- A reference is deleted via garbage collection after any names bound to it have passed out of scope
- Python uses *reference semantics* (more later)

Naming Rules

- *Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.*

```
bob  Bob  _bob  _2_bob_  bob_2  BoB
```

- *There are some reserved words:*

```
and, assert, break, class, continue, def, del,
elif, else, except, exec, finally, for, from,
global, if, import, in, is, lambda, not, or,
pass, print, raise, return, try, while
```

Assignment

- You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

- This makes it easy to swap values

```
>>> x, y = y, x
```

- Assignments can be chained

```
>>> a = b = x = 2
```

Accessing Non-Existent Name

Accessing a name before it's been properly created (by placing it on the left side of an assignment), raises an error

```
>>> y

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

Sequence Types: Tuples, Lists, and Strings

1. **Tuple: ('john', 32, [CMSC])**
   - A simple *immutable* ordered sequence of items
   - Items can be of mixed types, including collection types
2. **Strings: "John Smith"**
   - *Immutable*
   - Conceptually very much like a tuple
4. **List: [1, 2, 'john', ('up', 'down')]**
   - *Mutable* ordered sequence of items of mixed types

Sequence Types 1

- Define tuples using parentheses and commas

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Define lists are using square brackets and commas

```
>>> li = ["abc", 34, 4.34, 23]
```

- Define strings using quotes (“, ‘, or “““).

```
>>> st = "Hello World"
>>> st = 'Hello World'
>>> st = """This is a multi-line
string that uses triple quotes."""
```

Sequence Types 2

- Access individual members of a tuple, list, or string using square bracket "array" notation
- *Note that all are 0 based…*

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]      # Second item in the tuple.
 'abc'

>>> li = ["abc", 34, 4.34, 23]
>>> li[1]       # Second item in the list.
 34

>>> st = "Hello World"
>>> st[1]    # Second character in string.
 'e'
```

Positive and negative indices

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Positive index: count from the left, starting with 0

```
>>> t[1]
'abc'
```

Negative index: count from right, starting with –1

```
>>> t[-3]
4.56
```

Slicing: return copy of a subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying *before* second.

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

Negative indices count from end

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

Slicing: return copy of a =subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Omit first index to make copy starting from beginning of the container

```
>>> t[:2]
(23, 'abc')
```

Omit second index to make copy starting at first index and going to end

```
>>> t[2:]
(4.56, (2,3), 'def')
```

Copying the Whole Sequence

- [:] makes a *copy* of an entire sequence

```
>>> t[:]
(23, 'abc', 4.56, (2,3), 'def')
```

- Note the difference between these two lines for mutable sequences

```
>>> l2 = l1 # Both refer to 1 ref,
            # changing one affects both
>>> l2 = l1[:] # Independent copies, two refs
```

The ‘in’ Operator

- Boolean test whether a value is inside a container:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- For strings, tests for substrings

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Be careful: the *in* keyword is also used in the syntax of *for loops* and *list comprehensions*

The + Operator

The + operator produces a *new* tuple, list, or string whose value is the concatenation of its arguments.

```
>>> (1, 2, 3) + (4, 5, 6)
 (1, 2, 3, 4, 5, 6)

>>> [1, 2, 3] + [4, 5, 6]
 [1, 2, 3, 4, 5, 6]

>>> “Hello” + “ ” + “World”
 ‘Hello World’
```

The * Operator

- The * operator produces a *new* tuple, list, or string that “repeats” the original content.

```
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)

>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]

>>> “Hello” * 3
‘HelloHelloHello’
```

If Statements

Correct if statement

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Indentation matters

```
a = 33
b = 200

if b > a:
print("b is greater than a")
```

```
  File "demo_if_error.py", line 4
    print("b is greater than a")
        ^
IndentationError: expected an indented block
```

If Statements

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Simple function: ex.py

```
"""factorial done recursively and iteratively"""
def fact1(n):
    ans = 1
    for i in range(2,n):
        ans = ans * n
    return ans

def fact2(n):
    if n < 1:
        return 1
    else:
        return n * fact2(n - 1)
```

Simple functions: ex.py

```
671> python
Python 2.5.2 …
>>> import ex
>>> ex.fact1(6)
1296
>>> ex.fact2(200)
788657867364790503552363213932185O7…000000L
>>> ex.fact1
<function fact1 at 0x902470>
>>> fact1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'fact1' is not defined
>>>
```

GitHub Repository

- Everything related to this training will be uploaded to this GitHub repository:
  - https://github.com/JoHussien/GitHubCampus-ML

Heading to the Labs

References

- https://www.csee.umbc.edu/courses/671/fall09/notes/python1.ppt
- https://www.w3schools.com/python/python_conditions.asp