Week 8 Tutorial: Fundamentals of Python Programming I

POP77001 Computer Programming for Social Scientists

Module website: tinyurl.com/POP77001

Naming conventions

It is a good practice to follow usual naming convention when writing code.

- Use **UPPER_CASE_WITH_UNDERSCORE** for named constants (e.g. variables that remain fixed and unmodified)
- Use **lower_case_with_underscores** for function and variable names
- Use **CamelCase** for classes (more on them later)

Extra: PEP 8 -- Style Guide for Python Code

Code layout

- Limit all lines to a maximum of 79 characters.
- Break up longer lines

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]

result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )

income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Extra: PEP 8 -- Style Guide for Python Code

Reserved words

There are 35 reserved words (keywords) in Python (as of version 3.9) that cannot be used as identifiers.

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

Source: Python keywords

Defining variables

- Assignment statement binds the variable name and an object.

Defining variables

- Assignment statement binds the variable name and an object.

In [1]:

```
x = 5 # Variable 'x' is bound to object 5 of type integer
```

Defining variables

- Assignment statement binds the variable name and an object.

In [1]:

```python
x = 5 # Variable 'x' is bound to object 5 of type integer
```

- The same object can have multiple names (⚠️ more on aliasing and copying below)

Defining variables

- Assignment statement binds the variable name and an object.

In [1]:

```
x = 5 # Variable 'x' is bound to object 5 of type integer
```

- The same object can have multiple names (⚠️ more on aliasing and copying below)

In [2]:

```
y = x
x += 3 # Note that x was overriden even with addition operation as inte
print(x)
print(y)
```

```
8
5
```

Defining variables

- Assignment statement binds the variable name and an object.

In [1]:

```
x = 5 # Variable 'x' is bound to object 5 of type integer
```

- The same object can have multiple names (⚠️ more on aliasing and copying below)

In [2]:

```
y = x
x += 3 # Note that x was overriden even with addition operation as inte
print(x)
print(y)
```

```
8
5
```

In [3]:

```
try = 5 # Watch out for reserved words
```

```
  Input In [3]
    try = 5 # Watch out for reserved words
        ^
SyntaxError: expected ':'
```

Strings

- String (`str`) - **immutable ordered** sequence of characters
- Immutable - individual elements cannot be modified
- Ordered - strings can be sliced (unlike in R)

Strings

- String (`str`) - **immutable ordered** sequence of characters
- Immutable - individual elements cannot be modified
- Ordered - strings can be sliced (unlike in R)

```
In [4]: s = 'test'
        s
```

```
Out[4]: 'test'
```

Strings

- String (`str`) - **immutable ordered** sequence of characters
- Immutable - individual elements cannot be modified
- Ordered - strings can be sliced (unlike in R)

```
In [4]:  s = 'test'
         s

Out[4]:  'test'

In [5]:  s[0] # slicing (indexing starts from 0!)

Out[5]:  't'
```

Strings

- String (`str`) - **immutable ordered** sequence of characters
- Immutable - individual elements cannot be modified
- Ordered - strings can be sliced (unlike in R)

In [4]:

```
s = 'test'
s
```

Out[4]:

```
'test'
```

In [5]:

```
s[0] # slicing (indexing starts from 0!)
```

Out[5]:

```
't'
```

In [6]:

```
s[0] = 'r' # immutability
```

```
---------------------------------------------------------------
-------------
TypeError                                 Traceback (most recen
t call last)
Input In [6], in <cell line: 1>()
----> 1 s[0] = 'r'

TypeError: 'str' object does not support item assignment
```

Some string methods

```
s.capitalize()
s.title()
s.upper()
s.lower()
s.find(some_string)
s.replace(one_string, another_string)
s.strip(some_string)
s.split(some_string)
s.join(some_list)
```

Extra: Python string methods

Method chaining

- Recall from the lecture that methods can be *chained*
- E.g. `s.strip().replace('--', '---').title()`
- It provides a shortcut (does not necessitate intermediate objects)
- However, it can reduce code legibility! 📜

Exercise 1: Working with strings

- Remove trailining whitespaces (before and after the sentence) in the string below;
- Replace all double whitespaces with one;
- Format it as a sentence with correct punctuation;
- Print the result.

Exercise 1: Working with strings

- Remove trailining whitespaces (before and after the sentence) in the string below;
- Replace all double whitespaces with one;
- Format it as a sentence with correct punctuation;
- Print the result.

In [7]:

```
# Exercise 1:

s = "   truth  can  only be  found in  one place:  the  code "

# Your code goes here
```

Lists

- List (`list`) - **mutable ordered** sequence of elements
- Mutable - individual elements can be modified
- Ordered - lists can be sliced (like strings)

Lists

- List (`list`) - **mutable ordered** sequence of elements
- Mutable - individual elements can be modified
- Ordered - lists can be sliced (like strings)

In [8]:

```python
l = [1, 2, 3]
l
```

Out[8]:

```
[1, 2, 3]
```

Lists

- List (`list`) - **mutable ordered** sequence of elements
- Mutable - individual elements can be modified
- Ordered - lists can be sliced (like strings)

```
In [8]:  l = [1, 2, 3]
         l
```

```
Out[8]:  [1, 2, 3]
```

```
In [9]:  l[1] # slicing
```

```
Out[9]:  2
```

Lists

- List (`list`) - **mutable ordered** sequence of elements
- Mutable - individual elements can be modified
- Ordered - lists can be sliced (like strings)

```
In [8]:  l = [1, 2, 3]
         l

Out[8]:  [1, 2, 3]

In [9]:  l[1] # slicing

Out[9]:  2

In [10]: l[1] = 7 # immutability
         l

Out[10]: [1, 7, 3]
```

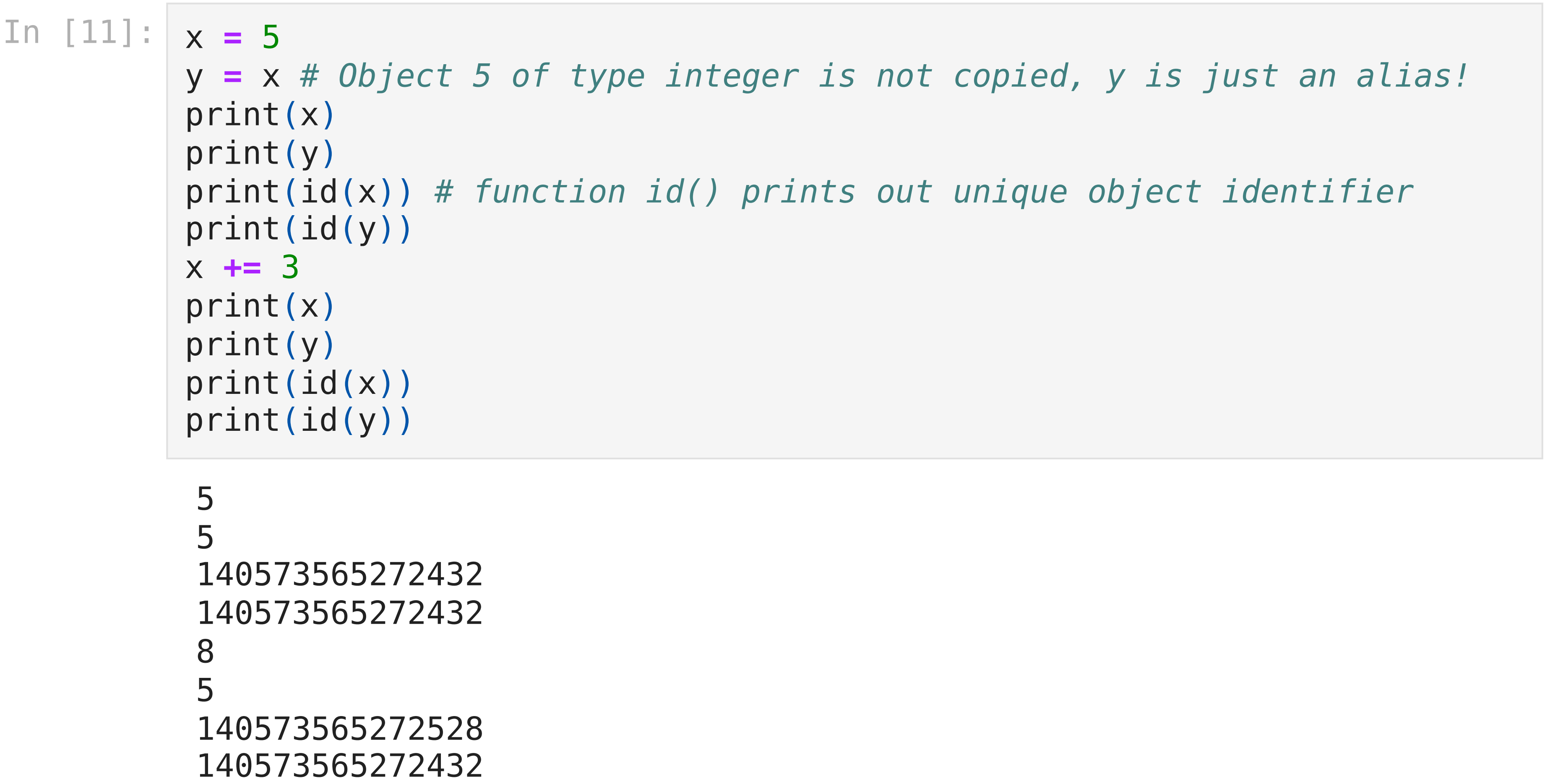
Some list methods

```
l.append(some_element)
l.extend(some_list)
l.insert(index, some_element)
l.remove(some_element)
l.pop(index)
l.sort()
l.reverse()
l.copy()
```

Extra: Python list methods

Aliasing vs copying

- As we saw above, the same object can have multiple names.
- It doesn't usually create a problem with immutable types, as the entire object just gets overriden
- But it might with mutable types!
- Compare below:



In [11]:

```
x = 5
y = x # Object 5 of type integer is not copied, y is just an alias!
print(x)
print(y)
print(id(x)) # function id() prints out unique object identifier
print(id(y))
x += 3
print(x)
print(y)
print(id(x))
print(id(y))
```

```
5
5
140573565272432
140573565272432
8
5
140573565272528
140573565272432
```

In [12]:

```
l1 = [1, 2, 3]
l2 = l1 # Object [1, 2, 3] of type list is not copied, l2 is just an al
l3 = l1[:]
l4 = l1.copy() # Both [:] slicing notation and copy method create copie
l2.pop(0) # Remove (and return) first element of the list
l3.insert(0, 0) # Insert 0 as the first element of the list
l4.append(4) # Append 4 to the end of the list
print(l1)
print(l2)
print(l3)
print(l4)
```

```
[2, 3]
[2, 3]
[0, 1, 2, 3]
[1, 2, 3, 4]
```

Exercise 2: Working with lists

- Below is a shuffled version of the first 11 elements of Fibonacci sequence.
- Create a copy of the shuffled list;
- Remove the last element;
- Sort it from smaller integers to larger;
- Select the second smallest and the third largest integers in the sequence; Print them out;
- Replace them in the list with the string, containing word corresponding to that number (e.g. 'two' for 2);
- Print out the results.

In [13]:

```
# Exercise 2:

fib_shuffled = [34, 5, 3, 1, 13, 55, 21, 2, 8, 0, 1]

# Your code goes here
```

Week 8 Exercise (unassessed)

- Practice working with built-in Python data structures