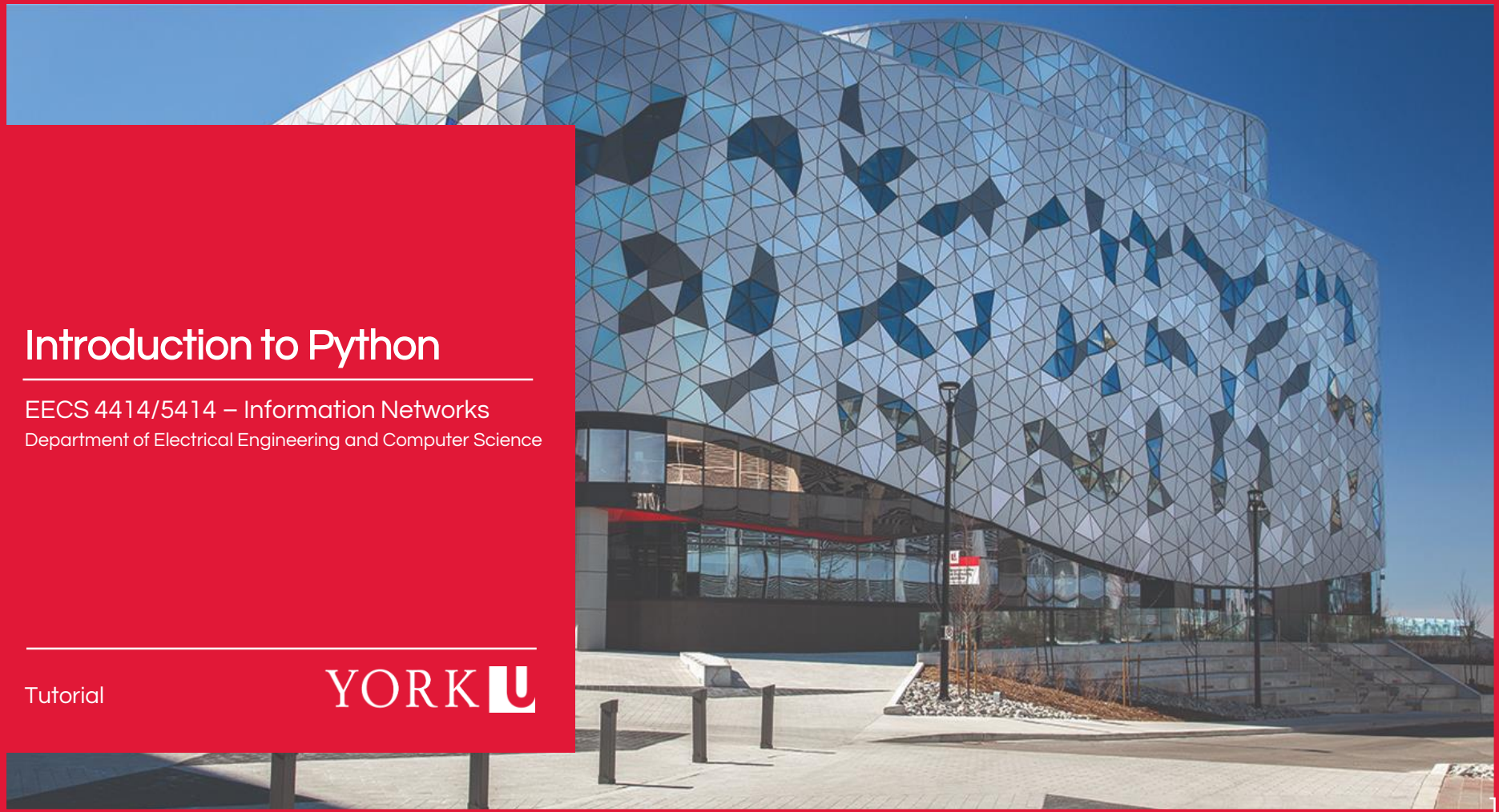
Introduction to Python

EECS 4414/5414 – Information Networks

Department of Electrical Engineering and Computer Science

Tutorial

YORK U



Why Python?

- A scripting language
- Easy to learn
- Object-oriented
- Numerous libraries, including tools for data analysis
- Powerful and scalable; has supporting tools that handle massive datasets

Simple Printing

Using the `print` function, and can handle a variety of data types (e.g., strings, numbers, booleans, etc.).

In [ ]:

```
print('Hello World!')
print(10)
print(True)
```

```
Hello World!
10
True
```

Python Syntax

- C-like
- Some exceptions
  - Missing operators: ++, --, etc.
  - No { } in blocks; use whitespace and indentation
  - Different keywords
  - Type declarations not needed
  - Other extra features!
- Use `#` to write comments.
- Run Python scripts:

```
$ python hello.py
```

Data Operators - 1

- Addition (`+`), Subtraction (`-`), Multiplication (`*`), Division (`/`), Integer Division (`//`), Modulus (`%`), Exponentiation (`**`)
- Does not have an increment (`++`) and decrement (`--`) operator; try instead `+=1` or `-=1`
- Assignment (`=`)
- Concatenation with strings (`+`)

Data Operators - 2

In [ ]:

```
print(3 + 8)
print(14 / 7)
print((10 % 3)**(2*100))

# working with variables and assignments
x = "bar"
x = 7 # reassignment to a new type is allowed
x += 1 # increment x by 1
print(x) # prints 8

# working with strings
y = "foo"
z = "apple"
print(y + z)
```

Data Types - 1

- **Number** - could either be an `int` a `float`, or even `complex`
- **Strings** - enclosed by quotation marks (characters are single-lengthed strings)
- **Boolean** – a `True` or `False`
- **Lists**

In [ ]:

```
a = [1, 2, 3, 4, 5]
print (a[1]) # 2
some_list = []
some_list.append("foo")
some_list.append(12)
print (len(some_list)) # 2
```

Data Types - 2

- Tuples
  - Lists are mutable but tuples are not
  - Lists can expand but tuples cannot
  - Tuples are slightly faster

In [ ]:

```
t = (10, 20, 30)
print(t[2]) # prints 30

# Here's a trick to add elements to your tuple
# Add 40 to the end of t to make it a 4-tuple
t2 = t + (40,)
print(t2)   # prints (10, 20, 30, 40)
```

Data Types - 3

- **Dictionaries**
  - Consists of key-value pairs
  - Keys have to be unique
  - Can access a dictionary's value based on the supplied key

In [ ]:

```
a = {“age”: 18, “b”: “123a”, 3: True}
print (a[3])
print (a[“age”])
```

```
True
18
```

Control Flow – Conditional Statements

In [ ]:

```
grade = 95
if grade > 90:
   print('Excellent')
elif grade > 80:
   print('Average')
elif grade > 70:
   print('Passing')
else:
   print('Needs Improvement')
```

```
Excellent
```

Control Flow – Loops

- These two codes are virtually the same!

In [ ]:
```
num = 0
while num < 5:
    print(num)
    num += 1
```

In [ ]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Control Flow – Loops and Lists

- Consider for each:

In [ ]:

```
L = [1,2,3,4,5]
L2 = []
for i in range(len(L)):
    power = 2**L[i]
    L2.append(power)
print(L2)                          # prints [2, 4, 8, 16, 32]
```

Control Flow – Loops and Lists

- Without indexing!

In [ ]:

```
L = [1,2,3,4,5]
L2 = []
for item in L:
    power = 2**item
    L2.append(power)
print(L2)                          # prints [2, 4, 8, 16, 32]
```

- Virtually the same thing!

Control Flow – Loops and Lists

- **Advanced**: use list comprehension!

In [ ]:

```
L = [1,2,3,4,5]
L2 = [2**x for x in L]
print(L2)                    # prints [2, 4, 8, 16, 32]
```

- Virtually the same thing!
- Much shorter!

Functions

- All variables are `local` unless specified as `global`
- Arguments are passed by value

In [ ]:

```
def test_method(k):
    return k**2 + 7

print(test_method(10))   # output 107
```

```
107
```

Modules

- Some libraries are not built-in when installing Python
- Can import modules using `import` ; sometimes in conjunction with `from` depending on the scope of how much of the package you need
- The * can be useful to denote all.

In [ ]:
```
import math
print(math.sqrt(2.0))
```

```
1.4142135623730951
```

In [ ]:
```
from math import sqrt
print (sqrt(2.0))
```

```
1.4142135623730951
```

Resources

- Visit my Github repository for introduction and notebooks of Python and NetworkX
  - https://github.com/techGIAN/NetworkX-Tutorial
- Google Colab examples:
  - https://colab.research.google.com/drive/1WYCdobWIcabI8ZUUfBgy8uAgSst8LBiS?usp=sharing
- Python tutorials:
  - https://www.w3schools.com/python/
  - https://www.learnpython.org/
- Python documentation:
  - https://docs.python.org/3/

Thank you!

Questions?