INFO3180 - LECTURE 1

# PYTHON ESSENTIALS

PYTHON IS POWERFUL... AND FAST;
PLAYS WELL WITH OTHERS;
RUNS EVERYWHERE;
IS FRIENDLY & EASY TO LEARN;
IS OPEN.

python.org

## OVERVIEW

- Python is a widely used programming language.
- It's also popular for web-based applications (e.g. web application frameworks such as Flask and Django)
- It comes already installed on MacOS and Linux. You will need to install it on Windows.
- We will be using Python 2 (NOT Python 3)

## VARIABLES

## EXAMPLES

```
id = 12345
myString = 'Hello World'
isValid = True
```

## FUNCTIONS AND INDENTATION

## FUNCTIONS AND INDENTATION

- Python functions have no explicit **`begin`** or **`end`**, and no curly braces to mark where the function code starts and stops. The only delimiter is a colon (**`:`**) and the indentation of the code itself.
- Indentation is very important in Python and can result in errors if code is not indented properly.
- It is recommended that you indent with ***4 spaces*** and **NO tabs**.

## EXAMPLE

```
def add(x, y):
    return (x + y)

print add(1, 2)

# output
# 3
```

## COMMENTS

## EXAMPLES OF COMMENTS

```
# This is a single line comment
```

```
'''
This is a
multi-line comment.
'''
```

## CONTROL FLOW

## IF/ELSE IF/ELSE

```python
if x < 0:
    x = 0
    print 'Negative changed to zero'
elif x == 0:
    print 'Zero'
elif x == 1:
   print 'Single'
else:
   print 'More'
```

## FOR STATEMENT

```
# Measure some strings:
words = ['cat', 'window', 'bigglesworth']
for w in words:
    print w, len(w)

# output would be
# cat 3
# window 6
# bigglesworth 12
```

## WHILE LOOP

```
def myFunction(n):
    a = 1
    while a < n:
        print a

myFunction(3)
# output
# 1
# 2
```

## RANGE

```
range(10)

# output
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


range(5, 10)
# output
# [5, 6, 7, 8, 9]
```

## INPUT/OUTPUT

## USER INPUT

- You can use the `input()` or `raw_input()` methods
- From the command line it will prompt the user to enter a value.

## EXAMPLES

```
x = input('What is your name?')

print 'Your name is: ' + x
```

## STRING FORMATTING

- You can use the `format()` method (recommended)
- Or you can use "%" operator.

## EXAMPLES

```
'hello world, my name is {0} and my id number is
{1}'.format('Yannick', '12345')
```

```
'hello world, my name is %s and my id number is %s' %
('Yannick','620012345')
```

## DATA STRUCTURES

## LISTS

- Similar to arrays in other languages.
- Defined by square brackets "`[]`" and values are separated with commas.

## LIST COMPREHENSION

- Provide a concise way to create lists.
- Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence.
- List comprehension consists of brackets containing an expression followed by a `for` clause, then zero or more `for` or `if` clauses.

## EXAMPLE

```
numbers = [1, 2, 3, 4, 5]
doubled_odds = []

for n in numbers:
    if n % 2 == 1:
    doubled_odds.append(n * 2)

doubled_odds

# output
[2, 6, 10]
```

```
# The above can be shorted to the following:
numbers = [1, 2, 3, 4, 5]
doubled_odds = [n * 2 for n in numbers if n % 2 == 1]
```

## TUPLES

- Similar to lists but they are immutable (unable to be changed).
- Are defined using parentheses "**( )**" and comma separated values.

## DICTIONARIES

- Similar to lists, however, they are not indexed numerically.
- They function similar to associative arrays or hashes in other languages.
- Defined using curly braces "**{ }**" and comma separated values.

## EXAMPLES

```python
# List
x = ['say', 'what', 3, 'times']

# Tuple
y = (1, 2, "hello")

# Dictionary
z = {'name': 'John', 'age': 10, "gender": "male",
"awesome": True}
```

## CLASSES

## EXAMPLE CLASS

```
class MyClass:
    """A simple example class"""
    id = 12345

    def __init__(self, name):
        self.name = name

    def say_message(self):
        return 'hello world, my name is {0} and my id 
number is {1}'.format(self.name, self.id)

student = MyClass('Yannick')
print student.say_message()
"""hello world, my name is Yannick and my id number is 
12345"""
```

## MODULES

## MODULES

- Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module.
- definitions from a module can be imported into other modules or into the main module
- A module is a file containing Python definitions/functions and statements.
- The file name is the module name with the suffix `.py` appended. e.g. if the file name is `fibo.py` then the module name would be `fibo`.

## EXAMPLE

```
# Fibonacci numbers module in a file called fibo.py

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b
        a, b = b, a+b

# …other function definitions

# You can then import the whole module into another file
# or on the command line
import fibo
fibo.fib(500)

# or you could import specific items
from fibo import fib, foo, baz
fib(500)
```

**Now go forth and *conquer!***

## RESOURCES

- Python Official Website - http://python.org
- Python Official Docs - https://docs.python.org/2/
- Codecademy Python Lessons - https://www.codecademy.com/learn/python
- Learn Python - https://www.learnpython.org/
- List Comprehensions: Explained Visually - http://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/