# COMP3160: Artificial Intelligence
# Python Primer: Part B

Rolf Schwitter
Rolf.Schwitter@mq.edu.au

# Today's Agenda

- Modules
- Files I/O
- Classes
- Regular Expressions

# Modules

- A module allows you to logically organize your Python code.
- Basically, a module is a file consisting of Python code.

```
# Import module math
>>> import math

# Now you can call a defined function in that module
# as follows:
>>> math.sqrt(3)
1.732050807568772
```

# Modules

- ***from X import a, b, c*** imports the module ***X***, and creates references in the current namespace to the given objects.
- You can now use ***a*** and ***b*** and ***c*** in your program.

```
>>> from math import factorial, pi, sqrt
>>> factorial(3)
6
>>> pi
3.141592653589793
>>> sqrt(3)
1.7320508075688772
```

# The `dir()` Function

- The `dir()` function returns all properties and methods of the specified object.

```
>>> import math
>>> names = dir(math)
>>> names
['__doc__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

# Text File: unit-description.txt

Artificial Intelligence (AI) is a well-established field that studies how computers and computer software capable of exhibiting intelligent behaviour can be designed. In this unit students will be exposed to fundamental concepts in AI such as agent architecture, knowledge representation, planning and search, as well as their application in some topical domains. Upon completion of this unit students will be able to apply problem-solving strategies that are required to build intelligent systems.

# Reading Files

```
# Read lines
file = open("unit-description.txt", "r")
print(file.readlines())
file.close()

# Read entire text without line breaks; no need to call
# file.close() when using a with statement.
with open("unit-description.txt", "r") as file:
    print(file.read())

# Loop over lines
file = open("unit-description.txt", "r")
for line in file:
    print(line)
file.close()
```

# Writing Files

```
# Write lines to file.
file = open("test.txt", "w")
file.write("This is a test.\n")
file.write("Add more lines.")
file.close()

# Just to check the result:
file = open("test.txt", "r")
for line in file:
    print(line)
file.close()
```

# Exceptions

- Syntax:

```
# try:
#    You do your operations here;
#    ......................
# except ExceptionI:
#    If there is ExceptionI, then execute this block.
# except ExceptionII:
#    If there is ExceptionII, then execute this block.
#    ......................
# else:
#    If there is no exception then execute this block.
```

# Exceptions

```
while True:
    try:
        n = input("Please enter an integer: ")
        n = int(n)
        break
    except ValueError:
        text = """You did not enter an integer: {}
Please try again.""".format(ValueError)
        print(text)

print("Great, you successfully entered an integer!")
```

# Classes

- Classes encapsulate variables and functions into a single entity.
- Objects get their variables and functions from classes.
- A class is basically a blueprint to create objects (instances).

```
class Person:
    def __init__(self, name, age):
    self.name = name        # Class attributes are variables
    self.age = age          # self is used to represent the
                            # instance of a class.

p1 = Person("John", 36)
print(p1.name)              # Access attribute of an instance
print(p1.age)               # using the dot notation.
```

# Classes

- The method `__init__()` gets called whenever a new object of the class is instantiated.
- This method is used to assign values to object properties or other operations required when the object is created.
- Any variable prefixed with `self` indicates that the variable is available to every method in the class. It is used to access variables that belong to the class.
- The name does not need to be self, but it has to be the first parameter of any function in the class.
- Objects can contain methods (= functions that belong to the object).

# Example: Classes

```
class Shape:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y

    def perimeter(self):
        return 2 * self.x + 2 * self.y
```

# Example: Classes

```
def describe(self, text):
    self.description = text

def authorName(self, text):
    self.author = text

def scaleSize(self, scale):
    self.x = self.x * scale
    self.y = self.y * scale
```

# Using Classes

```
# Creating an instance:
rectangle = Shape(100, 45)

# Finding the area of your rectangle:
print(rectangle.area())

# Finding the perimeter of your rectangle:
print(rectangle.perimeter())

# Describing the rectangle:
rectangle.describe("A wide rectangle, more than twice\
 as wide as it is tall")

# Reducing x and y by 50%:
rectangle.scaleSize(0.5)

# Printing new x and y:
print(rectangle.x)
print(rectangle.y)
```

# Exercise

Define a class **Person** together with two child classes: **Male** and **Female** in Python. Each classes should have a method **get_gender** which can print **male** for the **Male** class and **female** for the **Female** class.

Show how you can instantiate the two child classes by declaring a variable for each class and how you print the gender for the resulting instances.

# Solution

```
class Person(object):
    def get_gender(self):
        return "unknown"

class Male(Person):
    def get_gender(self):
        return "male"

class Female(Person):
    def get_gender(self):
        return "female"

a_male   = Male()
a_female = Female()
print(a_male.get_gender())
print(a_female.get_gender())
```

# Features and Tricks

```
>>> # Unpacking
>>> a, b, c = [1, 2, 3]
>>> a, b, c
(1, 2, 3)
>>> # Generator
>>> [i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # Help
>>> help(str)
Help on class str in module builtins:
class str(object)
 |  str(object='') -> str
...
```

# Features and Tricks

```
>>> # Flatten a list in one line: list comprehension
>>> list = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
>>> flat_list = [item for sublist in list for item in sublist]
>>> flat_list
[1, 2, 3, 4, 5, 6, 7, 8, 9]

# flat_list = [item for sublist in list for item in sublist]
# means:
# >>> flat_list = []
# >>> for sublist in list:
#         for item in sublist:
#           flat_list.append(item)
```

# Features and Tricks

```
>>> # Dictionary comprehension
>>> {x: x**3 for x in range(10)}
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343,
8: 512, 9: 729}

>>> # Set comprehension
>>> nums = {n**2 for n in range(10)}
>>> nums
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}

>>> # Lambda function
>>> x = lambda a, b : a * b
>>> print(x(3, 2))
6
```

# Pip Install

- Pip is a Python tool to install and manage packages.
- Pip is installed as a default in the newest versions of Python.
- For an overview of the available commands and syntax for Pip, use: `C:>pip help`
- For example, the following command installs the NLTK toolkit: `C:>pip install nltk`

# Regular Expressions

- A regular expression is an object that describes a pattern of characters.
- Regular expressions are used to perform pattern-matching and search-and-replace functions on text.
- You can find the regular expression syntax that is available in Python here:

  **`https://www.tutorialspoint.com/python/python_reg_expressions.htm`**
- Classic regular expressions can be compiled into Deterministic Finite Automata that can match a string of length n in O(n) time.

# Regular Expressions in Python

```
>>> import re
>>> str1 = "parrot monkey parrot"
>>> mo = re.match(r'parrot', str1)
>>> mo.group(0)
'parrot'
>>> str2 = "a parrot monkey parrot"
>>> mo = re.match(r'parrot', str2)
>>> mo.group(0)
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    mo.group(0)
AttributeError: 'NoneType' object has no attribute 'group'
>>>
```

# Regular Expressions in Python

```
>>> so = re.search(r'parrot', str2)
>>> so.group(0)
'parrot'
>>> so.start()
2
>>> so.end()
8
>>> res = re.findall(r'parrot', str2)
>>> res
['parrot', 'parrot']
```

# Regular Expressions in Python

```
>>> contactInfo = "Doe, John: 555-1212"
>>> # \w+ = one or more word character
>>> so = re.search(r'(\w+), (\w+): (\d{3}-\d{4})', contactInfo)
>>> so.group(0)
'Doe, John: 555-1212'
>>> so.group(1)
'Doe'
>>> so.group(2)
'John'
>>> so.group(3)
'555-1212'
```

# Regular Expressions in Python

- Creating a new regular expression pattern to match many strings can be slow in Python.
- Therefore, it is recommended to compile regular expressions if one needs to test or extract information from many input strings using the same expression.
- For example:

```
>>> regex = re.compile(r"(\w+) World")
>>> text = "Hello World, Bonjour World"
>>> for result in regex.findall(text): print(result)

Hello
Bonjour
```

# Take-Home Messages

- Modules are part of a code library and contain a set of functions to include in your code.
- A class is a "blueprint" for creating objects.
- A regular expression is a special sequence of characters that helps to match or find strings or sets of strings.
- Regular expressions are most powerful in sorting and data validation (including form inputs).
- Regular expressions are often difficult to understand.
- Do not use regular expressions when a parser exists.