Fall 2016

Faculty of Engineering
Computer Engineering Department
CMP 402

Machine Intelligence
Python – Part 1

Objectives:

By the end of this session, students should be able to:

- Define different basic types of variables int, float, bool, string, list
- Write expressions and use some built-in functions that operate on different class types of variables.
- Use print() to output the defined variables
- Know how to write single line and multi-line comments
- Define more complex types: two-dimensional lists, tuple, dictionary

Preparation steps:

- Download the appropriate version at https://www.python.org/downloads/
- Download aima-python master from https://github.com/aimacode/aima-python
- Create a Python project in eclipse
- Add the files in the lab folder variables_expressions.py, agents.py – the file in the lab folder is a modified version of the original one in aima-python folder- (It imports utility and grid so utility.py and grid.py should be added as well from the aima-python folder)

Sample code: variables_expressions.py

```
#int and float variables
x = 8
y = 20

print(x)
print("Addition: "+str(x+y)) #
expression evaluation needs explicit
type conversion to string
print("Multiplication: "+str(x*y))
print("Division: "+str(y/x))
print("Floor Division: "+str(y//x))
print("2 Power 8: "+str(2**8))
print("y Modulus 3: "+str(y%3))
k = 4.0
print("x times k: "+str(x*k))

print(type(x))
print(type(k))
print(type(y/x))
print(type(y//x))
```

```
8
Addition: 28


Multiplication: 160
Division: 2.5
Floor Division: 2
2 Power 8: 256
y Modulus 3: 2

x times k: 32.0

<class 'int'>
<class 'float'>
<class 'float'>
<class 'int'>
```

```
#bool
a= True
b = False
print(type(a))

print("a AND b : "+str(a and b))
print("a OR b : "+str(a or b))
print("NOT a  : "+str(not a))

print("a bitwise AND b : "+str(a & b))
print("a bitwise OR b : "+str(a | b))
print("a XOR b : "+str(a ^ b))

print("9 > 5 : "+ str(9>5) )
print("9 < 5 : "+ str(9<5) )
print("9 == 8 : "+ str(9==8) )
print("9 != 8 : "+ str(9!=8))
```

```
<class 'bool'>

a AND b : False
a OR b : True
NOT a  : False

a bitwise AND b : False
a bitwise OR b : True
a XOR b : True

9 > 5 : True
9 < 5 : False
9 == 8 : False
9 != 8 : True
```

```
#string
z= 'hello'
l= "world"
print(type(z))
```

```
<class 'str'>
```

```
name = "class 2017"
concatenated_str = "{} {}
{}!".format(z.capitalize(),l.capitalize(
),name.capitalize())
print(concatenated_str)
print(concatenated_str.split(sep=' '))
print(concatenated_str.split(sep=' ')[2]
[:-1])
print(concatenated_str.replace('Class
2017', '2017 Graduates'))
print(concatenated_str.find("Class"))
```

```
Hello World Class 2017!
['Hello', 'World', 'Class', '2017!']
Class
Hello World 2017 Graduates!

12
```

```
print(concatenated_str+' didn\'t
change')
```

```
Hello World Class 2017! didn't change
```

```
text= concatenated_str+"\n" "Hi
"+name.capitalize()
print(text[:23])
print(text)
print('C:\some\name')  # here \n
oncatenated_str.split(sep=' ')means
newline!
print(r'C:\some\name') # r: raw string
to avoid interpreting special characters
print(3*'Hi')
print(3*'Hi'.swapcase())
print(3*'Hi'.casefold())
print(3*'Hi'.upper())
print(text.partition('\n'))

print(text.partition('\n')[2])

print(text.splitlines())


print(' '.join(text.splitlines()))
print(' '.join([z,l,name]))
```

```
Hello World Class 2017!
Hello World Class 2017!
Hi Class 2017

C:\some
ame
C:\some\name

HiHiHi
hIhIhI
hihihi
HIHIHI
('Hello World Class 2017!', '\n', 'Hi
Class 2017')
Hi Class 2017

['Hello World Class 2017!', 'Hi Class
2017']

Hello World Class 2017! Hi Class 2017
hello world class 2017
```

```
numbers=[1,13,301,5343,89380]
for n in numbers:
    print(str(n).zfill(5))
```

```
00001
00013
00301
05343
89380
```

```
#List
squares = [1,4,8,16,25]
print(type(squares))
print(type(squares[1]))
print(squares[2]) #indexing
squares[2] = 9 #mutable
```

```
<class 'list'>
<class 'int'>
```

```
print(squares[:])
print(squares[-2]) #circular list
squares.append(36)
print(squares)
squares+= [49,64,81,100]
print(squares)
print(len(squares))
squares[10:11]=[121,144]
print(squares[:6])
print(squares[6:])



#2d list
matrix = [[1,2,3],[4,5,6],[7,8,9]]
print(matrix)
print(type(matrix))
print(matrix[0])
matrix[1][0]=0
print(matrix[1])
matrix.append([10,11,12])
print(matrix)
print(len(matrix))
print(len(matrix[3]))
```

```
8


[1, 4, 9, 16, 25]
16

[1, 4, 9, 16, 25, 36]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
10

[1, 4, 9, 16, 25, 36]
[49, 64, 81, 100, 121, 144]



[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
<class 'list'>
[1, 2, 3]
[0, 5, 6]



[[1, 2, 3], [0, 5, 6], [7, 8, 9], [10,
11, 12]]
4
3
```

```
#tuple
location =(0,10)
print(location)
print(type(location))
(x,y)=location
print(x)
print(y)
location=(x,y,20)
print(location)
print(location.count(0))
print(location.__getitem__(2))
print(location.index(20))
```

```


(0, 10)
<class 'tuple'>

0
10

(0, 10, 20)
1
20
2
```

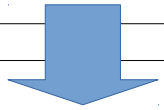
```
#dict: dictionary
french = dict()
french['yes'] = 'oui'
french['no'] = 'non'
french['one'] = 'un'
french['two'] = 'deux'
french['three'] = 'trois'
print(french)
print(type(french))
print(french['two'])
french.__setitem__("four", "quatre")
print("Count in french: {one}, {two},
{three}, {four}".format(**french))
```

```


{'yes': 'oui', 'no': 'non', 'one': 'un',
'three': 'trois', 'two': 'deux'}
<class 'dict'>
deux
Count in french: un, deux, trois, quatre
```

Listing 1.1: variables_expressions.py

MI AIMA code: agents.py

```
print("Reflex Vacuum Agent :")
print("====================")
a = ReflexVacuumAgent()
print(a.program((loc_A, 'Clean')))#'Right'
print(a.program((loc_B, 'Clean')))#'Left'
print(a.program((loc_A, 'Dirty'))) #'Suck'
print(a.program((loc_B, 'Dirty'))) #'Suck'

e = TrivialVacuumEnvironment()
e.add_thing(a,(0,0))
print(e.percept(a))
print(a.program(e.percept(a)))
e.execute_action(a, a.program(e.percept(a)))
print(e.percept(a))
print(a.program(e.percept(a)))
e.execute_action(a, a.program(e.percept(a)))
print(e.percept(a))
```



```
Reflex Vacuum Agent :
====================
Right
Left
Suck
Suck
((0, 0), 'Dirty')
Suck
location : (0, 0) , status: {(1, 0): 'Dirty', (0, 0): 'Dirty'}
 === > action: Suck , performance=10
((0, 0), 'Clean')
Right
location : (0, 0) , status: {(1, 0): 'Dirty', (0, 0): 'Clean'}
 === > action: Right , performance=9
((1, 0), 'Dirty')
```

Listing 1.2: Reflex Vacuum Agent (agents.py)

```
print("Model Based Vacuum Agent :")
print("==========================")
e2 = TrivialVacuumEnvironment()
a2= ModelBasedVacuumAgent()
e2.add_thing(a2)
e2.run(steps=5)
```

```
Model Based Vacuum Agent :
==========================
location : (0, 0) , status: {(1, 0): 'Dirty', (0, 0): 'Dirty'}
 === > action: Suck , performance=10
location : (0, 0) , status: {(1, 0): 'Dirty', (0, 0): 'Clean'}
 === > action: Right , performance=9
location : (1, 0) , status: {(1, 0): 'Dirty', (0, 0): 'Clean'}
 === > action: Suck , performance=19
location : (1, 0) , status: {(1, 0): 'Clean', (0, 0): 'Clean'}
 === > action: NoOp , performance=19
location : (1, 0) , status: {(1, 0): 'Clean', (0, 0): 'Clean'}
 === > action: NoOp , performance=19
```

Listing 1.3: Model based Vacuum Agent

```
print("Table Driven Based Vacuum Agent :")
print("==========================")
e3 = TrivialVacuumEnvironment()
e3.status = {loc_A: 'Dirty', loc_B:  'Dirty'}
a3= TableDrivenVacuumAgent()
e3.add_thing(a3)
e3.run(steps=5)
```

```
Table Driven Based Vacuum Agent :
==========================
location : (1, 0) , status: {(1, 0): 'Dirty', (0, 0): 'Dirty'}
 === > action: Suck , performance=10
location : (1, 0) , status: {(1, 0): 'Clean', (0, 0): 'Dirty'}
 === > action: None , performance=10
location : (1, 0) , status: {(1, 0): 'Clean', (0, 0): 'Dirty'}
 === > action: None , performance=10
location : (1, 0) , status: {(1, 0): 'Clean', (0, 0): 'Dirty'}
 === > action: None , performance=10
location : (1, 0) , status: {(1, 0): 'Clean', (0, 0): 'Dirty'}
 === > action: None , performance=10
```

Listing 1.4: Table Driven Based Vacuum Agent

Practice

Please solve the following straight-forward exercises (estimated time: max 15 min)

1. Update `TableDrivenVacuumAgent` to get the correct expected actions highlighted in Listing 1.4 output
2. Write python code to split the below paragraph into sentences and print one sentence per line with capitalizing the first letter in sentence.
   "horses have joined a select group of animals that can communicate by pointing at symbols.scientists trained horses, by offering slices of carrot as an incentive, to touch a board with their muzzle to indicate if they wanted to wear a rug.the horses' requests matched the weather, suggesting it wasn't a random choice.a few other animals, including apes and dolphins, appear, like us, to express preferences by pointing at things”
3. Write python code that would replace all occurrences of number '1' in a string with one (e.g. *"I have 1 son and 1 daughter. We have 1 house and 1 car."* would print: `I have one son and one daughter. We have one house and one car.`
4. Write python code that the calculates the cube of a list of numbers (e.g. [3,5,10] would output [27,125,1000].