Python Study Notes

February 9, 2022

1 Python Basics

This notebook serves as the sutdy notes for Python Language. The material is mainly follow the MIT courseware Introduction to Computer Science and Programming Using Python. This Chapter describes the basic `objects/while/for/if/logic/operations` concepts in Python language.

- Scalar and Non-scalar Objects
- Expressions in Python
- Binding Variables
- Comparison Operators
- Logic Operators
- Conditional Statement
- Strings
- While Loops
- For Loops
- Iteration
- Guess and Check

1.1 Scaler and Non-scalar Objects

`int`, `float`, `bool`, `NoneType` are built-in scalar obejcts (Python is an object-oriented language and eveything in python is an object of a class).

`list`,`tuple`, `list`, `dictionary` are non-scalar objects.

Can use `type()` to see the type (class) of an object.

We can directly convert object of one type to another.

[ ]:
```
print(type(5))
```

[ ]:
```
print(type(3.0))
```

[ ]:
```
print(float(3))
print(int(3.9))
```

1.2 Expressions in Python

Syntax for a simple expression `<object> <operator> <object>`

Common operators on `ints` and `floats` are `+,-,*,/`, **int division** `//`, **remainder** `%` and the **power** `**`.

Parentheses have the highest priority.

[ ]:

```
# 6/3 get 2.0 returns float 3.0, 5//2 returns integer 2
print(6/3)
print(5//2)
print(5%2)
```

1.2.1 Input/Output in Python

Keywords are `print(), input()`

`Text = input("Type anything...")`, input() takes string input and we can convert string to integer using `num=int(input("Type a number"))`

1.3 Binding variables and Values

Equal sign is an assignment of a value to a variable name. Re-bind variable names using new assignment statements. Previous value may still stored in memory but lost the handle of it.

1.4 Comparion Operators

Used for integers and floats `i>j`,`i>=j`, `i<j`, `i<=j`, **equality test** `i==j`, and **inequality test** `i!=j`.

1.5 Logic Operators

Used for bools `not a`, `a and b`, `a or b`.

1.6 Conditional Statement

`if (conditon): ... elif (condition): ... else:`, indentation matters in Python

[ ]:

```
# x = int(input('Enter an interger')) # input in Python
# COMPOUND BOOLEANS
x = 1; y =2; z = 3
if x<y and x<z:
    print('x is least')
elif y<z: # ELSE IF
    print('y is least')
else:
    print('z is least')
```

1.7 Strings

Strings can represent letters, special characters, spaces, digits. Strings are enclosed in double or single quotation marks.

1. Double quotation is handy and we can mainly use double quotes.

2. Use + to add (concatenate) strings together
3. Use " " as a blank space
4. String is a **non-scalar** object, meaning there are attributes associated with each string object.

[ ]:
```python
hi = "Hello There!"
print(hi)
name = "eric!"
greeting = hi+" "+name
print(greeting)
```

1.7.1 String Operations

`Concatenation, successive concatenation, length, indexing, slicing, reverse, in` (Note python uses a 0-based indexing system, while MATLAB uses the 1-based indexing system).

Strings are **immutable**; however, we can do re-assignment to modify the String.

[ ]:
```python
hi = 'ab'+'cd' # CONCATENATION
print(hi)
hi1 = 3*'eric' # SUCCESSIVE CONCATENATION
print(hi1)
hi2 = len('eric') # THE LENGTH, ALSO INCLUDES THE SPACE
print(hi2)
hi3 = 'eric'[1] # INDEXING, BEGINS WITH INDEX 0, THIS RETURNS r
print(hi3)
hi4 = 'eric'[1:3] # SLICING, EXTRACTS SEQUENCE STARTING AT FIRST INDEX AND ENDING BEFORE THE 3 INDEX
print(hi4)

# STRING OPERATION EXAMPLES
str1 = 'hello'
str2 = ','
str3 = 'world'

print('a' in str3) # bool, False, in/not in ARE TWO BASIC PYTHON MEMBERSHIP OPERATORS
print('HELLO' == str1) # bool, False
str4 = str1 + str3 # STRING CONCATENATION
print('low' in str4) # bool, True
print(str3[:-1]) # string, worl, note -1 means the last element, -2 means the second last element
print(str4[1:9:2]) # string, elwr, EXTRACT THE LETTERS WITH INDEX 1,3,5,7
print(str4[::-1]) # string, dlrowolleh, (REVERSE ORDER)
print(str4) # str4 itself is not changed in slicing operations
s = "hello"
s = "y" + s[1:len(s)] # strings are immutable, but we can re-assign the string.
print(s)
```

1.7.2 String Comparison Operations

==, !=, >, >=, <, <=

PYTHON COMPARES STRING LEXICOGRAPHICALLY (USING ASCII VLAUE OF CHARACTERS)

e.g. Str1 = “Mary”, Str2 = “Mac”, THE FIRST TWO CHARS ARE M = M, THE SECOND CHARS ARE THEN COMPARED a,a

ARE STILL EQUAL, THE THIRD TWO CHARS ARE THEN COMPARED r(ASCII 114) > c (ASCII 099)

A<B<C<…<Z<a<b<c<…<x<y<z

[ ]:

```python
print("tim" == "tie") # False
print("free" != "freedom") # True
print("arrow" > "aron") # True
print("right" >= "left") # True
print("teeth" < "tee") # False
print("yellow" <= "fellow") # False
print("abc">"") # True, NOTE THE EMPTY STRING "" IS SMALLER THAN ALL OTHER STRINGS
```

1.7.3 String Method

1. EVERYTHING IN PYTHON IS AN OBJECT. OBJECTS ARE SPECIAL BECAUSE WE CAN ASSOCIATE SPECIAL FUNCTIONS, REFERRED TO AS OBJECT METHODS, WITH THE OBJECT.
2. More methods associated with Strings can be found here

[ ]:

```python
s = 'abc'
s.capitalize # returns the function type
s.capitalize() # invoke the function and returns Abc (need () to indicate a method is invoked)
print(s.capitalize())
s.upper() # Return a copy of the string with all the cased chars converted to uppercase
print(s.upper())
print(s.isupper()) # Return true if all cased characters in the string are uppercase
            # and there is at least one cased character, false otherwise.
print(s.islower()) # similar to s.isupper
print(s.swapcase()) #Return a copy of the string with uppercase chars converted to lowercase, vice versa.
print(s.find('e')) # Return the lowest index in the string where substring 'e' is found,-1 if sub is not found
print(s.index('c')) # Like find(), but raise ValueError when the substring is not found.
print(s.count('e')) # Return the number of non-overlapping occurrences of␣
 ↪substring e
print(s.replace('old','new')) # Return a copy of the str, all occurrences of␣
 ↪substr 'old' replaced by 'new'
```

1.8 While Loops

`while <condition>: <expression>`, note `<condition>` evaluates to a Boolean. If `<condition>` is `True`, do all the steps inside the while code block, and then check the `<condition>` again and repeat until `<condition>` is `False`.

Indentation matters!

[ ]:

```
# CONTRL FLOW while LOOPS , range(start,stop,step)
n = 0
while n<5: # CTRL + c IN THE CONSOLE TO STOP THE PROGRAM
    print(n)
    n = n+1
```

1.9 For Loops

`for n in range(5)`, is equivalent to n in [0,1,2,3,4]

`range(7,10)` starts at 7 stops at 10 (7,8,9) and `range(5,11,2)` starts at 5, stops at 11, step 2 (5,7,9)

`break` can be used for exiting the innermost loop (for,while)

`for` can loop through characters in strings

[ ]:

```
# break STATEMENT
mysum = 0
for i in range(5,11,2):
    mysum = mysum + i
    if mysum == 5:
        break
print(mysum)

# h, o ,l, a (for CAN LOOP CHARACTERS IN THE STRING)
for letter in 'hola':
    print(letter)
```

1.10 Iteration

Repeatedly use the same code. Need to set an iteration variable outside loop then test variable to determine when done and change variable within the loop.

Iterative algorithms allow us to do more complex things than simple arithmetic, one useful example are **guess and check** methods.

[ ]:

```python
x = 3
ans = 0
itersLeft = x
while(itersLeft != 0):
    ans = ans +x
    itersLeft = itersLeft - 1
print(str(x)+'*'+str(x)+'='+str(ans))
```

1.11 Guess and Check Algorithm

We guess a solution and check iteratively. Guess a value for solution. Check if the solution is correct. Keep guessing until find solution or guessed all values. The process is exhaustive enumeration. Can work on problems with a finite number of possibilities.

[ ]:

```python
# GUESS-AND-CHECK-cube root
cube = 28
for guess in range(abs(cube)+1):
    if guess**3 >= abs(cube):
        break
if guess**3 != abs(cube):
    print(cube, 'is not a perfect cube')
else:
    if cube < 0:
        guess = -guess
    print('Cube root of ' + str(cube) + ' is ' + str(guess))
```

2 Function/Iteration/Recursion/Modules/Files

This Chapter describes the Python `function/iteration/recrusion/modules/files`

- Bisection Search Algorithm
- Floats amd Fractions
- Newton-Rampson Root Finding Algorithm
- Functions
- Recursion
- Modules
- Files

2.1 Bisection Search Algorithms

We can use this algorithm to compute the monthly payment of a mortgage.

[ ]:

```python
"""
BISECTION SEARCH - SQUARE ROOT
# REALLY RADICALLY REDUCES COMPUTATION TIME
"""
x = 25
epsilon = 0.01
```

```
numGuesses = 0
low = 1.0
high = x
ans = (high + low)/2.0

while abs(ans**2-x) >= epsilon:
    print('low = '+str(low)+' high = '+str(high)+' ans = '+ str(ans))
    numGuesses += 1
    if ans**2 < x:
        low = ans
    else:
        high = ans
    ans = (high + low)/2.0

print('numGuesses = '+ str(numGuesses))
print(str(ans) + ' is close to square root of '+ str(x))
```

[ ]:

```
"""
BISECTION SEARCH - CUBE ROOT
# THIS SCRIPT ALSO ADDRESSES THE CASES WHERE X IN (-1,1) AND X < 0
"""
x = -8
epsilon = 0.01
numGuesses = 0
low = 1.0
high = abs(x)

if abs(x) <= 1:
    low = 0
    high = 1

ans = (high + low)/2.0 # BISECTION METHOD

while abs(ans**3-abs(x)) >= epsilon:
    print('low = '+str(low)+' high = '+str(high)+' ans = '+ str(ans))
    numGuesses += 1
    if ans**3 < abs(x):
        low = ans
    else:
        high = ans
    ans = (high + low)/2.0

if x < 0:
    ans = -ans

print('numGuesses = '+ str(numGuesses))
print(str(ans) + ' is close to cubic root of '+ str(x))
```

2.2 Floats and Fractions

1. Comupter represent numbers in binary format
2. Decimal number 302 = `3*100 + 0*10 + 2*1`
3. Convert an interger to binary form
4. For floats, IF WE MULTIPLE BY A POWER OF 2 (e.g 2^3) WHICH IS BIG ENOUGH TO CONVERT INTO A WHOLE NUMBER, CAN THEN CONVERT TO BINARY, AND THEN DIVIDE BY THE SAME POWER OF 2
    1. e.g. 3/8 = 0.375 = *310^-1 +* 710^-2 + 5*10^-3; 0.375*(2**3) = 3 (DECIMAL), THEN CONVERT TO BINARY (NOW 11)
    2. THEN DIVIDE BY 2**3(SHIFT RIGHT) TO GET 0.011 (BINARY)
5. THERE ARE SOME PORBLEMS WITH COMPRAING TWO FLOAT POINTS BECAUSE COMPUTER TRIES TO SEE IF THE BINARIES ARE SAME.
    1. WE ALWAYS USE abs(x-y)< some small number, rather than x == y

[ ]:
```
#THE FOLLOWING PROGRAM CONVERTS INTERGERS TO BINARY FORMS
num = -10
if num < 0:
    isNeg = True
    num = abs(num)
else:
    isNeg = False
result = ''
if num == 0:
    result = '0'
while num > 0:
    result = str(num%2) + result
    num = num//2
if isNeg:
    result = '-'+ result
print(result)
```

[ ]:
```
x = float(input('Enter a decimal number between 0 and 1:'))
p = 0
while ((2**p)*x)%1 != 0: # CONVERT TO A WHOLE NUMBER
    print('Remainder = '+str((2**p)*x-int((2**p)*x)))
    p += 1

num = int(x*(2**p))

result = ''
if num == 0:
    result = '0'
while num > 0: # CONVERT TO BINARY
    result = str(num%2) + result
    num = num//2

for i in range(p-len(result)):
    result = '0' + result

result = result[0:-p]+'.'+result[-p:]
print('The binary representation of the decimal '+str(x)+' is'+str(result))
```

2.3 Newton-Raphson

GENERAL APPROXIMATION ALGORITHM TO FIND ROOTS OF A POLYNOMIAL IN ONE VARIABLE P(X)=a_n x^n + a_n-1 x^n-1 + … + a_1 x + a_0 = 0

[ ]:

```
# WE USE THIS METHOD TO SOLVE p(x) = x^2 -24 = 0, WHREE x IS THE SQUARE ROOT OF 24
epsilon  = 0.01
y = 24.0
guess = y/2.0
numGuesses = 0

while abs(guess*guess - y) >= epsilon:
    numGuesses += 1
    guess = guess - (((guess**2)-y)/(2*guess)) # Intuitive explanation from wiki
print('numGuesses = '+str(numGuesses))
print('Square root of '+str(y)+' is about '+ str(guess))
```

2.4 Functions

1. Called/invokded/; `parameter/docstrings/body`; key word `def`; variable scope/global scope/function scope
2. Returns None if no return given
3. `printName(lastName = 'Huang',firstName = 'Zipeng', reverse = False)` (Most robust way with default value)

[ ]:

```
"""
FUNCTIONS
ARE NOT RUN IN A PROGRAM UNTIL THEY ARE CALLED/INVIKED
THEY HAVE: NAME, PARAMETERS (0, OR MORE), DOCSTRING(EXPLAIN WHAT A FUNCTION DOES) , BODY
"""
def is_even(i): # def IS A KEYWORD, IS_EVEN (NAME), i is PARAMETER/ARGUMENT
    """
    INPUT: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("hi")
    return i%2 == 0 # None, if no return given, only one return executed inside a function
```

```
                    # code insider function but after return statement noe excuted

x = is_even(3) # x is False

def func_a(): # no parameter
    print('inside func_a')

def func_b(y):
    print('inside func_b')
    return y

def func_c(z):
    print('inside func_c')
    return z()

print(func_a())
print(5+func_b(2))
print(func_c(func_a)) # call func_c, takes one parameter, another function (a func invokes another func)

# INSIDE A FUCNTION, CAN ACCESS A VARIABLE DEFINED OUTSIDE
# INSIDE A FUCNTION, CANNOT MODIFY A VARIABLE DEFINED OUTSIDE
def g(y):
    print(x)
    print(x+1) # x = x+1 is not valid
x = 5
g(x)
print(x)
```

[ ]:

```
"""
KWYWORD ARGUMENTS AND DEFAULT VALUES
"""
def printName(firstName,lastName,reverse):
    if reverse:
        print(lastName + ','+firstName)
    else:
        print(firstName,lastName)

# EACH OF RHESE ONVOCATIONS IS EQUIVALENT
printName('Zipeng','Huang',False)
printName('Zipeng','Huang',reverse = False)
printName('Zipeng',lastName = 'Huang',reverse = False)
# THE LAST INVOCATION IS RECOMMENDED SINCE IT IS ROBUST
printName(lastName = 'Huang',firstName = 'Zipeng', reverse = False)

"""
```

```
WE CAN ALSO SPECIFY THAT SOME ARGUMENTS HAVE DEFAULT VALUES, SO IF NO VALUE
SUPPLIED, JUST USE THAT VALUE  Default value
"""
def printName(firstName,lastName,reverse = False):
    if reverse:
        print(lastName + ','+firstName)
    else:
        print(firstName,lastName)

printName('Zipeng','Huang')
printName('Zipeng','Huang',True)
```

2.5 Recursion

1. DIVIDE AND CONQUER, A FUNCTION CALLS ITSELF (Mathematical Induction Reasoning)
    1. WE SOLVE A HARD PROBLEM BY BREAKING IT INTO A SET OF SUBPROBLEMS SUCH THAT:
    2. SUB-PROBLEMS ARE EASIER TO SOLVE THAN THE ORIGINAL
    3. SOLUTIONS OF THE SUB-PROBELMS CAN BE COMBINED TO SOLVE THE ORIGINAL
2. RECURSIVE STEP: THINK HOW TO REDUCE PROBLEM TO A SIMPLER/SMALLER VERSION OF SAME PROBLEM.
3. BASE CASE: KEEP REDUCING RPOBLEM UNTIL REACH A SIMPLE CASE THAT CAN BE SOLVED DIRECTLY.
4. ITERATION vs. RECURSION (DOES THE SAME THING)
    1. RECURSION MAY BE SIMPLER, MORE INTUITIVE
    2. RECURSION MAY BE EFFICIENT FROM PROGRAMMER'S POINT OF VIEW
    3. RECURSION MAY NOT BE EFFICIENT FROM COMPUTER POINT OF VIEW

[ ]:

```
# MULTIPLICATION-RECURSIVE SOLUTION
# MATHEMATICAL INDUCTION REASONING OF THE CODE
def mult(a,b):
    if b == 1: # BASE CASE
        return a
    else: # RECURSIVE STEP
        return a + mult(a,b-1)

# FACTORIAL
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)
```

[ ]:

```
# TOWERS OF HANOI (THINK RECURSIVELY! )
# SOLVE A SMALLER PROBELM/ SOLVE A BASIC PROBLEM
# fr: from stack (INITAL TOWER); to: to stack (FINAL TOWER); spare: (SPARE␣
 ↪TOWER);
def printMove(fr,to):
    print('move from'+str(fr)+'to'+str(to))

def Towers(n,fr,to,spare):
    if n==1:
        printMove(fr,to)
    else:
        # WE CAN HAVE MULTIPLE RECURSIVE CALLS INSIDE A FUNCTION
        # THINK IT RECURSIVELY
        Towers(n-1,fr,spare,to) # move the stack size of n-1 to a spare disc
        Towers(1,fr,to,spare) # move the bottom to the desired disc
        Towers(n-1,spare,to,fr) # move the n-1 back to the desired disc

print(Towers(3,'P1','P2','P3'))
```

[ ]:

```
# RECURSION WITH MULTIPLE BASE CASES
# FIBONACCI NUMBERS
def fib(x):
    """assumes x an int >=0, returns Fibonacci of x """
    if x == 0 or x ==1: # base cases
        return 1
    else:
        return fib(x-1)+ fib(x-2) # we have two recurisve functions calls in a␣
 ↪return
```

[ ]:

```
# RECURSION ON NON-NUMERICS (STRINGS)
def isPalindrome(s):
    def toChars(s): # convert string to all lower cases
        s = s.lower() # convert to lower case
        ans = ''
        for c in s: # remove all the punctuations/space
            if c in 'abcdefghijklmnopqrstuvwxyz':
                ans = ans + c
        return ans

    def isPal(s):
        if len(s)<= 1: # recursive base case
            return True
        else:
            return s[0] == s[-1] and isPal(s[1:-1]) # recursive step happens␣
 ↪here
                                                    # we compare the fisrt and␣
 ↪last letter then
                                                    # we convert the problem to␣
 ↪a smaller probelm
```

```
    return isPal(toChars(s))
```

2.6 Modules

A module is a `.py` file containing python definitions and statements

`import circle` (import a module), we can then use `circle.pi` to access a attribute/method inside a module

`from circle import *` 1. from the module `circle` import everything 2. we can then directly use `pi` variable defined in cirle without suing `circle.pi` 3. Need to make sure no name collision when use importing method

[ ]:

```
# the file circle.py contains
pi = 3.14159
def area(radius):
    return pi*(radius**2)
def circumference(radius):
    return 2*pi*radius

# then we can import and use this module
import circle
pi = 3   # can still define the pi in the shell
print(pi) # 3
print(circle.pi) # 3.14159, look for pi defined in the module
print(circle.circumference(3)) # 18.84953999

# if we don't want to refer to functions and vars by their module, and the names don't
# collide with other bindings, then we can use:
from circle import* # means from the module, import everything (denoted by the star sign)
print(pi)
print(area(3)) # we can refer them by calling their own name
```

2.7 Files

Python provides an operating-system independent means to access files, using a file handle.

`nameHandle = open('kids','w')` (open kids for write, r for read)

`name = input('Enter name: ')`

`nameHandle.write(name+ '\n')`

`nameHandle.close()`, (() means that we are referring to the close() function)

[ ]:

```
"""
WRITE/READ FILES
```

```
"""
nameHandle = open('kids','w') # 'kid': name of file; w: write command
for i in range(2):
    name = input('Enter name: ')
    nameHandle.write(name+ '\n')
nameHandle.close()

nameHandle = open('kids','r') # read
for line in nameHandle:
    print(line)
nameHandle.close()
```