Chapter2

April 1, 2021

Scientific Computation (MKP3303)

> R.U.Gobithaasan (2021). Scientific Computing, Lectures for Undergraduate Degree Program B.Sc (Applied Mathematics), Faculty of Ocean Engineering Technology, University Malaysia Terengganu. https://sites.google.com/site/gobithaasan/LearnTeach

Chapter 2: Numbers, Expressions and Functions
1. Integers, rational numbers and irrational numbers. 2. Floating point numbers 3. Complex numbers 4. Expressions, assignment statements, equalities, 5. Control Flow Statements 6. Functions (Python & Mathematics) 7. Overflow error, underflow error and rounding-off error 8. Developing your own module

References: - w3schools Online Materials - SciPi Lecture Notes - Robert Johansson, Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib (2019, Apress). - Donaldson Toby,Python: Visual QuickStart Guide (2008, Peachpit Press). - Tony Gaddis-Starting Out with Python,(2018,Global Edition-Pearson Education) - Robert Johansson August, Introduction to Scientific Computing in Python Continuum Analytics, (2015)

1 Integers, Rational numbers and Irrational numbers

2 Floating Point Numbers (Reals)

with Booleans & Strings as well.

[1]:
```
type(3), type(3.3), type('UMT')
```

[1]:
```
(int, float, str)
```

[2]:
```
#Assigning values to variables

a = 2
b = 3.3
c = 2/3 # division
d = 'z'
e = 'UMT'
```

```
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
```

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

[3]:
```
c
```

[3]:
```
0.6666666666666666
```

[4]:
```
f = False
g = True
print(type(f))
print(type(g))
```

```
<class 'bool'>
<class 'bool'>
```

[5]:
```
not g
```

[5]:
```
False
```

[6]:
```
type(None)
```

[6]:
```
NoneType
```

[7]:
```
None in { None,2}
```

[7]:
```
True
```

2.0.1 Integer remarks

$19.3^{-3.2}$

[8]:
```
19.3 ** -3.2 #scientific notation
```

[8]:
```
7.695141518235658e-05
```

[9]:
```
3.4e03 #multiply by 10 power of 3
```

[9]:
```
3400.0
```

[10]:
```
.34
```

[10]: 0.34

[11]:
```
print(10+(-4))
print(10+-4)
```

```
6
6
```

[12]:
```
# 1/0 #meant to show error message
```

[13]:
```
import sys #loading an external modul called sys
```

[14]:
```
sys.maxsize ##maximum
```

[14]: 9223372036854775807

[15]:
```
help(divmod)
```

```
Help on built-in function divmod in module builtins:

divmod(x, y, /)
    Return the tuple (x//y, x%y).  Invariant: div*y + mod == x.
```

[16]:
```
quotiont, remainder = divmod(10,3)
quotiont, remainder
```

[16]: (3, 1)

2.0.2 Rational Numbers

[17]:
```
from fractions import Fraction
# from module import function
```

[18]:
```
p1=Fraction(1,5)
p2=Fraction(3,5)
print(p1+p2)
print(p2-p1)
```

```
4/5
2/5
```

[19]:
```
p1
```

[19]:
```
Fraction(1, 5)
```

[20]:
```
p1.numerator
```

[20]:
```
1
```

[21]:
```
{p1.numerator, p1.denominator} # seperating them using builtin function in␣
↪"fractions"
```

[21]:
```
{1, 5}
```

[22]:
```
p3=Fraction('2.1')
print(p3)
```

```
21/10
```

2.1 Irrational numbers

examples include $\sqrt{2}$ and π. We usually express in the form of flaot numbers for computation.

[23]:
```
import math as m # math module
print(m.pow(2,1/2))
print(m.sqrt(2))
print(m.pi)
```

```
1.4142135623730951
1.4142135623730951
3.141592653589793
```

2.1.1 Casting: to specify a type on to a variable.

[24]:
```
y = 7
type(y)
```

[24]:
```
int
```

[25]:
```
y = 7
int(y), float(y), complex(y), oct(y), hex(y), str(y) # the aoutput is tuple of␣
↪various types of y
```

[25]:
```
(7, 7.0, (7+0j), '0o7', '0x7', '7')
```

3 Complex Numbers

[26]:
```
complex(2,3)
```

[26]:
```
(2+3j)
```

[27]:
```
h = 1+2j;
print(type(h))
```

```
<class 'complex'>
```

[28]:
```
h.real, h.imag # printing separately using built in function
```

[28]:
```
(1.0, 2.0)
```

run `?complex.conjugate` to read the documantation

[30]:
```
hConj=h.conjugate()
print(hConj)
```

```
(1-2j)
```

4 Expressions, assignment statements, equalities

4.0.1 Basic arithmetic operators

[31]:
```
print(3+7)
print(3-4)
print(3*4)
print(4/3) # answer convert to float even though the inputs are integers
print(4//3) #integer division
print(9%3) # remainder
print(2**5) #exponentiation
print(pow(2,4)) #using power function which is in math module
```

```
10
-1
12
1.3333333333333333
1
0
32
16
```

[32]:
```
a=3*3;  # semi-colon to supress output
```

[33]:
```
a
```

[33]: 9

4.0.2 Assigment and Expression

[34]:
```
mass = 55 # kg
acceleration = 30 # m/s^2

force = mass * acceleration # Newton =kgms^-2
print (force)
```

1650

Writing the following functions (explicitly)

$f1(a, x) = ax^2$

[35]:
```
a = 1
x = 3
f1 = a * (x**2)
print(f1)
```

9

[36]:
```
(a,x) = (1,3) # assigning multiple variables in a line
print( a * (x**2))
```

9

[37]:
```
a, x = 1, 2 # assigning multiple variables in a line
print( a * (x**2))
```

4

[38]:
```
a = x = 2 # assigning multiple variables WITH ONE VALUE
print( a * (x**2))
```

8

$f2(x, y, z) = x^2 + y^3 + \sqrt{z}$

[39]:
```
x= 1
y = 3
z = 4
f2 = x**2 + (y**3) + m.sqrt(z)
print(f2)
```

30.0

4.0.3 Equalities: Comparison

[40]:
```
5 < 2, 5 > 2, 6 >= 3*2, 6 == 3*2, 7 >= 3*2, 7 != 3*2,
```

[40]: (False, True, True, True, True, True)

[41]:
```
3 in {1,3}, 5 not in {2,4,6}
```

[41]: (True, True)

5 Control Flow Statements

[42]:
```
 2 in [1,2,3] # checking whether an element is in the list
```

[42]: True

5.0.1 If statement

[43]:
```
if 2 in [1,2,3]: print ("yes, 2 is in the list!")
```

```
yes, 2 is in the list!
```

[44]:
```
if 2 in [1,2,3]:
    print ("yes, 2 is in the list!")
```

```
yes, 2 is in the list!
```

[45]:
```
a,b = 0,1

if a:
    print(str(a) + " is true")
```

[46]:
```
if b: print(str(b) + " is true")
```

```
1 is true
```

[47]:
```
not b
```

[47]: False

[48]:
```
c = 5 # choose an odd number to print
if c % 2 != 0:
    print("c is an odd number")
```

```
c is an odd number
```

5.0.2 if-else

[49]:
```
c = 4 # choose an even number to print else statement
if c % 2 != 0:
    print("c is an odd number")
else:
    print("c is an even number")
```

```
c is an even number
```

5.0.3 if-elif-else

[50]:
```
marks = 60 # choose an even number to print else statement

if marks > 70:
    print("Good")

elif marks <= 69 and marks >= 40:
    print("Average")

else:
    print("bad")
```

```
Average
```

5.0.4 while loop

[51]:
```
n = 5
factorial = 1

while n > 0:

    factorial = factorial * n
    n = n - 1

print(factorial)
```

```
120
```

[52]:
```
n = 1

while n <= 12:
    print(n,":",n*n)
    n = n + 1
```

```
1 : 1
2 : 4
3 : 9
4 : 16
5 : 25
6 : 36
7 : 49
8 : 64
9 : 81
10 : 100
11 : 121
12 : 144
```

5.0.5 while-else loop

[53]:
```
n = 1

while n <= 12:
    if n % 2 == 0:
        print(n,":",n*n)

    n = n + 1

else:
    print("Done! I am not printing odd numbers!")
```

```
2 : 4
4 : 16
6 : 36
8 : 64
10 : 100
12 : 144
Done! I am not printing odd numbers!
```

[54]:
```
type(a)
```

[54]:
```
int
```

5.0.6 for loop

[55]:
```
mylist = {2,3,14,5}

for x in mylist:
    print (x)
```

```
2
3
5
14
```

[56]:
```
mylist = {2,3,14,5}
total=0

for element in mylist:
    total = total + element

print(total)
```

```
24
```

5.0.7 for-else loop

[57]:
```
mylist = {2,3,14,5}

for x in mylist:
    if x % 2 == 0:
        print (x)
else:
    print("Done! not interested with odd numbers")
```

```
2
14
Done! not interested with odd numbers
```

5.0.8 Range

[58]:
```
a = range(0,7)
print(a)
```

```
range(0, 7)
```

[59]:
```
for i in range(0,7):
    print(i)
```

```
0
1
2
3
4
5
6
```

[60]:
```
for i in range(0,20,5): # range(start,stop, stepsize)
    print(i)
```

```
0
5
10
15
```

5.0.9 skiping and exiting a part of the loop

[61]:
```
mylist = {2,3,14,5}

for x in mylist:
    if x % 2 == 0:
        print (x)
    else:
        continue # skipping odd numbers
```

```
2
14
```

[62]:
```
mylist = {2,3,14,5}

for x in mylist:
    if x % 2 == 0:
        print (x)
    else:
        break #breaking when element 3 is in the loop
```

```
2
```

6 Functions

[63]:
```
height = int(input('Enter your height: '))
```

```
Enter your height:  516
```

[64]:
```
height
```

[64]: 516

6.1 Mathematical Functions

[65]:
```
abs(-1), max(3,4,22.3), min(3,4,22.3)
```

[65]:
```
(1, 22.3, 3)
```

[66]:
```
import cmath
cmath.sqrt(-1)
```

[66]:
```
1j
```

[67]:
```
import math
math.ceil(5.3), math.floor((5.3)), math.exp(1), math.sqrt(2)
```

[67]:
```
(6, 5, 2.718281828459045, 1.4142135623730951)
```

[68]:
```
import random as rd
rd.random(), rd.randrange(6,10), rd.uniform(-2,0)
```

[68]:
```
(0.8673895931381005, 7, -1.1080238839487833)
```

6.2 User defined functions

6.2.1 Function returning no value

[69]:
```
def message():
    '''prints a welcoming message'''
    print("Selamat Datang Ke UMT")
```

[70]:
```
print(help(message))
```

```
Help on function message in module __main__:

message()
    prints a welcoming message

None
```

[71]:
```
def QEven(num):
    '''
    Query on even number:
    PRINTS a boolean output for an as integer": true if even number, otherwise␣
 ↪false'''
    print(num % 2 == 0)
```

[72]:
```
help(QEven)
```

```
Help on function QEven in module __main__:

QEven(num)
    Query on even number:
    PRINTS a boolean output for an as integer": true if even number, otherwise
false
```

[73]:
```
a = QEven(3)
b = QEven(40)
c = QEven(3.2)

print(a)
```

```
False
True
False
None
```

6.2.2 Function returning value(s)

[74]:
```
def QBEven(num):
    '''
    Query on even number:
    RETURNS a boolean output for an as integer": true if even number, otherwise␣
 ↪false'''
    return num % 2 == 0
```

[75]:
```
a = QBEven(6);

print(a)
```

```
True
```

[76]:
```
def SplitReal(num):
    nominator = int(num)
    denominator = num - nominator
    return int(nominator), float(denominator)
```

[77]:
```
a, b  = SplitReal(3.2)
print("Integer:",a)
print("Real:",b)
```

```
Integer: 3
Real: 0.20000000000000018
```

6.2.3 Representing mathematical function with def:

$f2(x, y, z) = x^2 + y^3 + \sqrt{z}$

[78]:
```
def f2(x,y,z):
    from math import sqrt
    return x**2 + y**3 + sqrt(z)

f2(2,3,4)
```

[78]: 33.0

6.3 Function with Global Variable

[79]:
```
def Add1(x,y,z):
    global sum2
    sum2 = x + y
    totalsum = sum2 + z
    return totalsum
```

[80]:
```
print(Add1(2,4,6))
print(sum2)
```

```
12
6
```

[81]:
```
def Add2(x,y,z):
    sum3 = x + y
    sum = sum3 + z
    return sum
```

[82]:
```
print(Add2(2,4,6))
#print(sum3)
```

```
12
```

6.3.1 Recursive Function

[83]:
```
def Factorial(x):
    if x == 0:
        return 1
    else:
        return (x) * Factorial(x-1)
```

[84]:
```
Factorial(5)
```

[84]: 120

6.3.2 Lambda Expression: simple function definition

[85]:
```
def Even1(num):return num % 2 == 0
```

[86]:
```
Even2 = lambda num:num % 2 == 0
```

[87]:
```
Even1(5), Even2(5)
```

[87]: (False, False)

7 Errors, rounding, overflow

Read Floating point aritmetic for more details.

[96]:
```
import math as m
print(format(m.pi, '.2g')) # give 12 significant digits
print(format(m.pi, '.2f'))  # give 2 digits after the point
print(repr(m.pi))
```

```
3.1
3.14
3.141592653589793
```

> the order is no more than 1 part in 2^{53} per operation on most machines, you may go up to 2^{56}

$0.1 \approx \frac{3602879701896397}{2^{55}}$

[116]:
```
3602879701896397 / (2 ** 55.) #the actual stored value is the nearest␣
↪representable binary fraction.
```

[116]: 0.1

[129]:
```
print(format(1/10, '.56g'))
```

```
0.10000000000000000555111512312578270211815834045410156 25
```

[130]:
```
0.1 == 0.10000000000000000555111512312578270211815834045410156 25
```

[130]: True

[146]:
```
print(format(0.1 + 0.1, '.56g'))
```

```
0.20000000000000001110223024625156540423631668090820312 5
```

[147]:
```
0.2==0.2000000000000000111022302462515654042363166809082031 25
```

[147]:
```
True
```

[145]:
```
print(format(0.1 + 0.1 + 0.1, '.56g'))
```

```
0.30000000000000004440892098500626161694526672363281 25
```

[144]:
```
0.3==0.30000000000000004440892098500626161694526672363281 25
```

[144]:
```
False
```

[97]:
```
0.1 + 0.1 + 0.1 == 0.3
```

[97]:
```
False
```

[151]:
```
from fractions import Fraction
Fraction(1/10) == 0.1000000000000000055511151231257827021181583404541015625
```

[151]:
```
True
```

[153]:
```
Fraction.from_float(0.1)
```

[153]:
```
Fraction(3602879701896397, 36028797018963968)
```

[154]:
```
Fraction.from_float(0.1) + Fraction.from_float(0.1) + Fraction.from_float(0.1)␣
↪== Fraction.from_float(0.3)
```

[154]:
```
False
```

[152]:
```
round(.1 + .1 + .1, 5) == round(.3, 5) #precision in 5 decimal digits.
```

[152]:
```
True
```

7.0.1 Overflow

Integers and Rational numbers do not overflow, however real numbers in the form of floats can!

[101]:
```
import sys

i = sys.maxsize
print(i)

print(i == i + 1) # MAX INTEGER + 1

i += 1
print(i)
```

```
9223372036854775807
False
9223372036854775808
```

Float can overflow!

[102]:
```
f = sys.float_info.max
print(f)

print(f == f + 1)  # MAX FLOAT + 1

f += 1
print(f)
```

```
1.7976931348623157e+308
True
1.7976931348623157e+308
```

8 Developing your own Module: simplified version

Module is a collection of functions/data that you have tested and saved as one python file with extension ".py". Hence, you may load (similar to those external modules) in your working cell to access all the functions/data that you have saved in this file.

Below is a (overly) simplified example.

MyFirstMNodule.py

```
def Greet(name):
    print("Apa Khabar " + name + "!")


def QBEven(num):
    """
    Query on even number:
    RETURNS a boolean output for an as integer": true if even number, otherwise false"""
    return num % 2 == 0


def Factorial(x):
    if x == 0:
        return 1
    else:
        return (x) * Factorial(x-1)
```

[103]:
```
import myModule
myModule.Greet("Minah")
```

```
Apa Khabar Minah!
```

[104]:
```
myModule.Factorial(8), myModule.QBEven(723)
```

[104]:
```
(40320, False)
```

[105]:
```
import myModule as G # renaming your module
G.Greet("Kevin")
G.Factorial(4)
```

```
Apa Khabar Kevin!
```

[105]:
```
24
```

[106]:
```
del myModule, G ## deleting loaded module or function
```

[107]:
```
from myModule import Factorial as fact #renaming a function
fact(4)
```

[107]:
```
24
```

You need to **reload** if you edit your module, otherwise the initial module will be executed.

[108]:
```
import importlib, myModule
importlib.reload(myModule)
```

[108]:
```
<module 'myModule' from '/Volumes/GoogleDrive/My Drive/0teaching/2021-2020/Sem-2
/MKP3303/ScientificComputingWithPython/NotebookLectures/myModule.py'>
```

[109]:
```
print(dir()) # check loaded module
```

```
['Add1', 'Add2', 'Even1', 'Even2', 'Factorial', 'Fraction', 'In', 'Out',
'QBEven', 'QEven', 'SplitReal', '_', '_1', '_10', '_100', '_104', '_105',
'_107', '_108', '_14', '_16', '_19', '_20', '_21', '_24', '_25', '_26', '_28',
'_3', '_33', '_40', '_41', '_42', '_47', '_5', '_54', '_6', '_64', '_65', '_66',
'_67', '_68', '_7', '_78', '_8', '_84', '_87', '_88', '_89', '_9', '_90', '_91',
'_92', '_93', '_94', '_95', '_97', '_98', '_99', '__', '___', '__builtin__',
'__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__',
'_dh', '_i', '_i1', '_i10', '_i100', '_i101', '_i102', '_i103', '_i104',
'_i105', '_i106', '_i107', '_i108', '_i109', '_i11', '_i12', '_i13', '_i14',
'_i15', '_i16', '_i17', '_i18', '_i19', '_i2', '_i20', '_i21', '_i22', '_i23',
'_i24', '_i25', '_i26', '_i27', '_i28', '_i29', '_i3', '_i30', '_i31', '_i32',
'_i33', '_i34', '_i35', '_i36', '_i37', '_i38', '_i39', '_i4', '_i40', '_i41',
'_i42', '_i43', '_i44', '_i45', '_i46', '_i47', '_i48', '_i49', '_i5', '_i50',
'_i51', '_i52', '_i53', '_i54', '_i55', '_i56', '_i57', '_i58', '_i59', '_i6',
'_i60', '_i61', '_i62', '_i63', '_i64', '_i65', '_i66', '_i67', '_i68', '_i69',
'_i7', '_i70', '_i71', '_i72', '_i73', '_i74', '_i75', '_i76', '_i77', '_i78',
'_i79', '_i8', '_i80', '_i81', '_i82', '_i83', '_i84', '_i85', '_i86', '_i87',
'_i88', '_i89', '_i9', '_i90', '_i91', '_i92', '_i93', '_i94', '_i95', '_i96',
'_i97', '_i98', '_i99', '_ih', '_ii', '_iii', '_oh', 'a', 'acceleration', 'b',
```

```
'c', 'cmath', 'd', 'e', 'element', 'exit', 'f', 'f1', 'f2', 'fact', 'factorial',
'force', 'g', 'get_ipython', 'h', 'hConj', 'height', 'i', 'importlib', 'm',
'marks', 'mass', 'math', 'message', 'myModule', 'mylist', 'n', 'p1', 'p2', 'p3',
'quit', 'quotiont', 'rd', 'remainder', 'sum2', 'sys', 'total', 'x', 'y', 'z']
```

[110]:
```
myModule.Greet("Minah")
```

```
Apa Khabar Minah!
```

[111]:
```
#del Factorial
#Factorial(3)
```

`Import` loads the whole module, but we may opt to load a specific module

[112]:
```
from myModule import Factorial
```

[113]:
```
print(Factorial(4))
```

```
24
```

[115]:
```
!jupyter nbconvert Chapter2.ipynb --to latex
```

```
[NbConvertApp] Converting notebook Chapter2.ipynb to latex
[NbConvertApp] Writing 96596 bytes to Chapter2.tex
```

[ ]:
```

```