CALCULUS FOR IT - 501031

Lab 1 manual – Python tutorial

We will use the Python language for all assignments in this course. In this Laboratory, we will introduce basic features of Python via common calculations and algorithms.

1. What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is used for web development (server-side), software development, mathematics, system scripting.

2. Python install

We can download Python for free from the following website (the latest version at this time is *3.9.7*): https://www.python.org/downloads/

To check if you have python installed on:

- a Windows PC:

  Search in the start bar for Python or run the following on the Command Line (cmd.exe), and type:

```
C:\Users\Your Name>python --version
```

- a Linux or Mac:

  Open the command line (Linux) or the Terminal (Mac), and type:

```
python --version
```

3. Python Quickstart

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

The content of the file "helloworld.py" is only one line:

```
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file (for ex., in the drive D:\), and run:

```
D:\>python helloworld.py
```

The output is:

```
Hello, World!
```

4. Python Indentation

Indentation refers to the spaces at the beginning of a code line. The indentation in Python is very important, and it indicate a block of code.

- Example:

```
if 5 > 2:
  print("Five is greater than two!")
```

- Python will give you an error if you skip the indentation:

```
if 5 > 2:
print("Five is greater than two!")
```

The content of error is: *IndentationError: expected an indented block*

- The number of spaces is up to you as a programmer, but it has to be at least one.

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

```
if 5 > 2:
 print("Five is greater than two!")
       print("Five is greater than two!")
```

The content of error is: *IndentationError: unexpected indent*

5. Python Comments

Comments can be used to explain Python code, make the code more readable, or prevent execution when testing code.

- Comments starts with a #, and Python will ignore them:

```
#This is a comment
print("Hello, World!")
```

```
print("Hello, World!") #This is a comment
```

- Multi Line Comments

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

6. Python Variables

- Creating Variables

  Python has no command for declaring a variable.

  A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```

- Casting

  If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

- Variable Names

  Rules for Python variables:

  -

  Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

  Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

  Multi Words Variable Names:

  Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:

  - Camel Case

    Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

  - Pascal Case

    Each word starts with a capital letter:

```
MyVariableName = "John"
```

  - Snake Case

    Each word is separated by an underscore character:

```
my_variable_name = "John"
```

- Global Variables

  Variables that are created outside of a function (as in all of the examples above) are known as global variables. Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

7. Python Data Types

Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

You can get the data type of any object by using the **type**() function:

```
x = 5
print(type(x))
```

8. Python Numbers

There are three numeric types in Python: int, float, and complex.

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
x = 35e3
y = 12E4
z = -87.7e100
```

Complex numbers are written with a "j" as the imaginary part:

```
x = 3+5j
y = 5j
z = -5j
```

You can convert from one type to another with the int(), float(), and complex() methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)
```

9. Python Booleans

Booleans represent one of two values: **True** or **False**.

In programming you often need to know if an expression is True or False. When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

10.Python Operators

- Python Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

- Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

- Python Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

- Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

11.Python Lists

- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

- List items are ordered, changeable, and allow duplicate values.
- To determine how many items a list has, use the **len**() function:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

- A list can contain different data types:

```
list1 = ["abc", 34, True, 40, "male"]
```

- List items are indexed and you can access them by referring to the index number (the first item has index **[0]**, the second item has index [1]):

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- To add an item to the end of the list, use the **append**() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- To insert a list item at a specified index, use the **insert**() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

- The **remove**() method removes the specified item:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

- The **pop**() method removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

12.Python Conditions and If statements

- Python supports the usual logical conditions from mathematics, for ex., a == b, a != b, a > b, … These conditions can be used in several ways, most commonly in "if statements" and loops.

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

- The **elif** keyword is pythons way of saying "if the previous conditions were not true, then try this condition":

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

- The **else** keyword catches anything which isn't caught by the preceding conditions:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

- The **and/or** keyword is a logical operator, and is used to combine conditional statements:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

- You can have if statements inside if statements, this is called nested if statements:

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

13.Python loops

- With the while loop we can execute a set of statements as long as a condition is true:

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

- With the **break** statement we can stop the loop even if the while condition is true:

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

- With the **continue** statement we can stop the current iteration, and continue with the next:

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

- A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string):

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

- To loop through a set of code a specified number of times, we can use the range() function:
  - The **range**() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

  - The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):
  print(x)
```

  - The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

14.Python Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

- A function is defined using the **def** keyword:
- To call a function, use the function name followed by parenthesis:

```
def my_function():
  print("Hello from a function")

my_function()
```

- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

- To let a function return a value, use the **return** statement:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

15.Python Math

Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

15.1 Built-in Math Functions

- The **min**() and **max**() functions can be used to find the lowest or highest value in an iterable:

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

- The **abs**() function returns the absolute (positive) value of the specified number:

```
x = abs(-7.25)

print(x)
```

- The **pow**(*x*, *y*) function returns the value of x to the power of y (x^y).

```
Return the value of 4 to the power of 3 (same as 4 * 4 * 4):

x = pow(4, 3)

print(x)
```

15.2 The Math Module

Python has also a built-in module called math, which extends the list of mathematical functions. To use it, you must import the **math** module:

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

Math Methods:

Method	Description
math.acos()	Returns the arc cosine of a number
math.acosh()	Returns the inverse hyperbolic cosine of a number
math.asin()	Returns the arc sine of a number
math.asinh()	Returns the inverse hyperbolic sine of a number
math.atan()	Returns the arc tangent of a number in radians
math.atan2()	Returns the arc tangent of y/x in radians
math.atanh()	Returns the inverse hyperbolic tangent of a number
math.ceil()	Rounds a number up to the nearest integer
math.comb()	Returns the number of ways to choose k items from n items without repetition and order
math.copysign()	Returns a float consisting of the value of the first parameter and the sign of the second parameter
math.cos()	Returns the cosine of a number
math.cosh()	Returns the hyperbolic cosine of a number
math.degrees()	Converts an angle from radians to degrees
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf()	Returns the error function of a number
math.erfc()	Returns the complementary error function of a number
math.exp()	Returns E raised to the power of x
math.expm1()	Returns E^x - 1

math.fabs()	Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor()	Rounds a number down to the nearest integer
math.fmod()	Returns the remainder of x/y
math.frexp()	Returns the mantissa and the exponent, of a specified number
math.fsum()	Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)
math.gamma()	Returns the gamma function at x
math.gcd()	Returns the greatest common divisor of two integers
math.hypot()	Returns the Euclidean norm
math.isclose()	Checks whether two values are close to each other, or not
math.isfinite()	Checks whether a number is finite or not
math.isinf()	Checks whether a number is infinite or not
math.isnan()	Checks whether a value is NaN (not a number) or not
math.isqrt()	Rounds a square root number downwards to the nearest integer
math.ldexp()	Returns the inverse of math.frexp() which is x * (2**i) of the given numbers x and i
math.lgamma()	Returns the log gamma value of x
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10()	Returns the base-10 logarithm of x
math.log1p()	Returns the natural logarithm of 1+x
math.log2()	Returns the base-2 logarithm of x
math.perm()	Returns the number of ways to choose k items from n items with order and without

	repetition
math.pow()	Returns the value of x to the power of y
math.prod()	Returns the product of all the elements in an iterable
math.radians()	Converts a degree value into radians
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
math.sin()	Returns the sine of a number
math.sinh()	Returns the hyperbolic sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh()	Returns the hyperbolic tangent of a number
math.trunc()	Returns the truncated integer parts of a number

Math Constants:

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

16.

17.References

- Python Tutorial on the W3schools website: https://www.w3schools.com/python/default.asp
-

-- THE END --