Introduction to Python Programming I

Theo Narh (Ph.D)

Graduate School of Nuclear & Allied Sciences (SNAS)
University of Ghana

Welcome!

UNIVERSITY OF GHANA

Important tool in this work

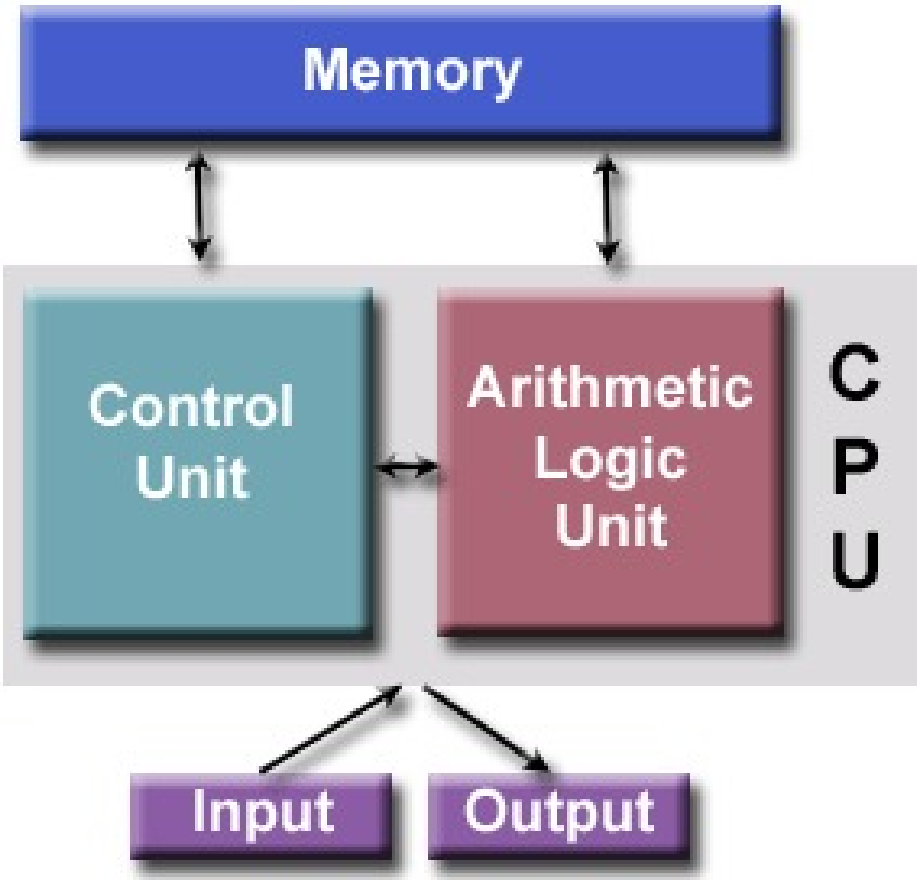
UNIVERSITY OF GHANA

Outline

UNIVERSITY OF GHANA

Basic Computer Design & Python installation

Introduction I



Basic Machine Architecture
Image credit: Livermore Computing Centre

- **Input/ Output** is the interface to the human operator.
- **Control unit** fetches instructions/ data from memory, decodes the instructions and then sequentially coordinates operations to accomplish the programmed task.
- **Arithmetic Unit** performs basic arithmetic operations.

UNIVERSITY OF GHANA

Introduction II

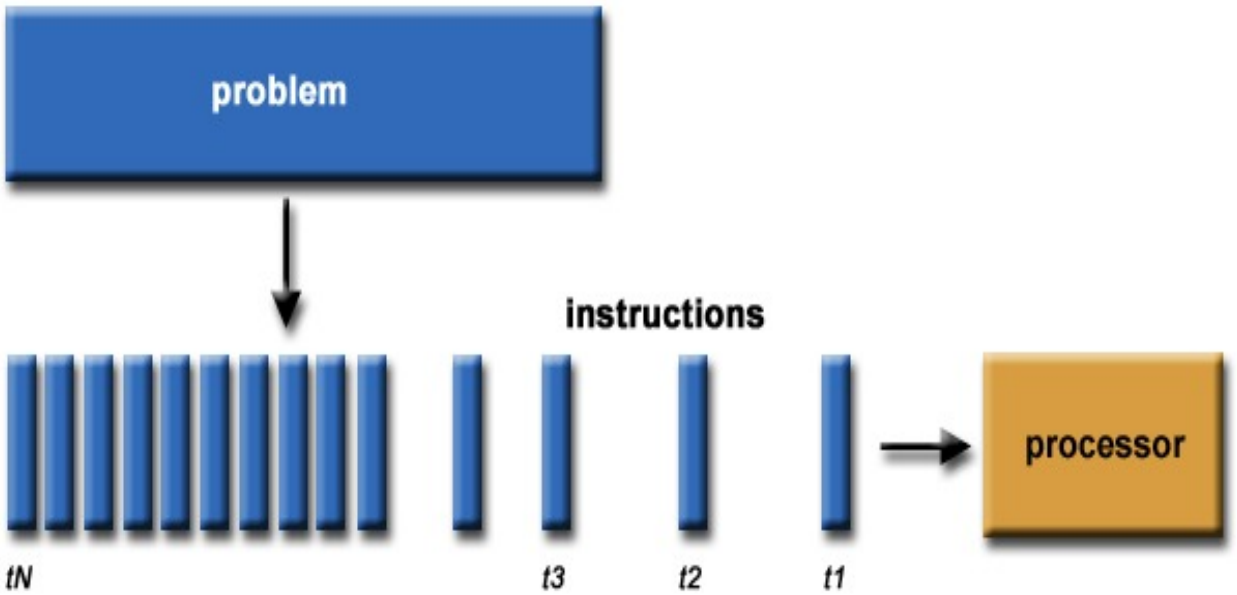
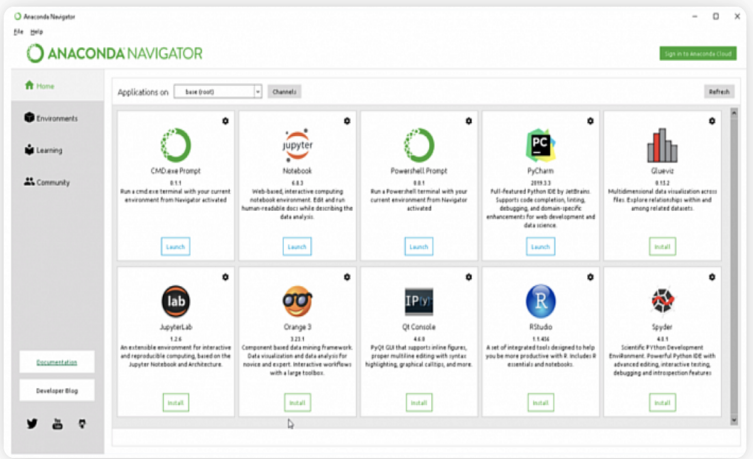


Figure 1: Serial Computing. *Credit*: Livermore Computing Centre

Introduction III



User interface makes learning easier

Anaconda Navigator is a desktop GUI that comes with Anaconda Individual Edition. It makes it easy to launch applications and manage packages and environments without using command-line commands.

Expedite your data science journey with easy access to training materials, documentation, and community resources including Anaconda.org.

Install Anaconda

Figure 2: Install **Anaconda** using the download link

UNIVERSITY OF GHANA

A simple program

```
print("Welcome to ICT lecture 1!")
```

Simple computations I

Arithmetic operators:

- + addition
- - subtraction
- * multiplication
- / division
- ** exponentiation
- % modulus

Compute: $1 + 2^3$

```
a = 1 + 2 ** 3
print(a)
```

Simple computations II

- Consider a simple formula for calculating the interest on a bank deposit: $A = P[1 + (r/100)^n]$. For instance, let $P = 100$, $r = 5.0$, and $n = 7$

```
100*(1 + 5.0/100)**7
# ----------------------

primary = 100
r = 5.0
n=7
amount = primary * (1+r/100)**n
print(amount)
```

- **NB:** A variable in a Python program can be any word containing the letters **a–z**, **A–Z**, underscore _ and the digits **0-9**, but it **cannot** start with a digit.

Simple computations III

- Comment your code:

```
# program for computing the growth of
# money deposited in a bank
primary = 100 # initial amount
r = 5.0                  # interest rate in %
n=7                       # the number of years
amount = primary * (1+r/100)**n
print(amount)
```

Data Types in Python

Data Types in Python I

- integers
- long integers
- floats (like doubles in C or Java)
- strings
- lists
- tuples
- dictionaries

1. **Integers**
   - Integers are numbers with no decimal parts, such as -5, -4, -3, 0, 5, 7 etc.
   - To declare an integer in Python, simply write variableName = initial value
   - Example: userAge = 20, mobileNumber = 12398724

UNIVERSITY OF GHANA

Data Types in Python II

2. **Float**
   - Float refers to numbers that have decimal parts, such as 1.234, -0.023, 12.01.
   - To declare a float in Python, we write variableName = initial value
   - Example: userHeight = 1.82, userWeight = 67.2
3. **String**
   - String refers to a text.
   - To declare a string, you can either use variableName = 'initial value' (single quotes) or variableName = "initial value" (double quotes)
   - Example:
     userName = 'Peter', userSpouseName = "Janet", userAge = '30'
     In the last example, because we wrote userAge = '30', userAge is a string. In contrast, if you wrote userAge = 30 (without quotes), userAge is an integer.

UNIVERSITY OF GHANA

Data Types in Python III

- We can combine multiple substrings by using the concatenate sign (+). For instance,

  ```
  "Peter" + "Lee"
  ```

  is equivalent to the string “PeterLee”.
- Built-In String Functions
  An example of a function available in Python is the upper() method for strings. You use it to capitalize all the letters in a string. For instance,

  ```
  'Peter'.upper()
  ```

  will give us the string “PETER”.
- Formatting Strings using the % Operator ::
  There are three parts to this syntax. First we write the string to be formatted in quotes. Next we write the % symbol. Finally, we have a pair of round brackets ( ) within which we write the values or variables to be inserted into the string. Example,

Data Types in Python IV

```
1  brand = 'Apple'
2  exchangeRate = 5.635235245
3  message = 'The price of this %s laptop is %d GH S and
   ↪  the exchange rate is %4.2f GH S to 1 USD' %(brand,
   ↪  20000, exchangeRate)
4
5  print(message)
```

- Formatting Strings using the format() method::
  In addition to using the % operator to format strings, Python also provides us with the format() method to format strings. The syntax is:

```
1  brand = 'Apple'
2  exchangeRate = 5.635235245
3  message = 'The price of this {0:s} laptop is {1:d} GH S
   ↪  and the exchange rate is {2:4.2f} GH S to 1
   ↪  USD'.format(brand, 20000, exchangeRate)
4
5  print(message)
```

UNIVERSITY OF GHANA

Data Types in Python V

- Type Casting In Python::
  Sometimes in our program, it is necessary for us to convert from one data type to another, such as from an integer to a string. This is known as type casting.

  There are three built-in functions in Python that allow us to do type casting. These are the int(), float(), and str() functions..

  The int() function in Python takes in a float or an appropriate string and converts it to an integer. To change a float to an integer, we can type int(5.712987). We'll get 5 as the result (anything after the decimal point is removed). To change a string to an integer, we can type int ("4") and we'll get 4. However, we cannot type int ("Hello") or int ("4.22321"). We'll get an error in both cases.

Data Types in Python VI

The float() function takes in an integer or an appropriate string and changes it to a float. For instance, if we type float(2) or float("2"), we'll get 2.0. If we type float("2.09109"), we'll get 2.09109 which is a float and not a string since the quotation marks are removed.

The str() function on the other hand converts an integer or a float to a string. For instance, if we type str(2.1), we'll get "2.1".

4. List
   - List refers to a collection of data which are normally related. Instead of storing these data as separate variables, we can store them as a list. For instance, suppose our program needs to store the age of 5 users. Instead of storing them as user1Age, user2Age, user3Age, user4Age and user5Age, it makes more sense to store them as a list.
   - To declare a list, you write listName = [initial values]. Note that we use square brackets [ ] when declaring a list.

Data Types in Python VII

Multiple values are separated by a comma. Example:

```
userAge = [21, 22, 23, 24, 25]
```

- We can also declare a list without assigning any initial values to it. We simply write

  ```
  listName = []
  ```

  What we have now is an empty list with no items in it. We have to use the append() method mentioned below to add items to the list.
- Individual values in the list are accessible by their indexes, and indexes always start from **ZERO**, not **1**. This is a common practice in almost all programming languages, such as C and Java. Hence the first value has an index of 0, the next has an index of 1 and so forth. For instance, userAge[0] = 21, userAge[1] = 22
- Alternatively, you can access the values of a list from the back. The last item in the list has an index of -1, the second last has an index of -2 and so forth. Hence, userAge[-1] = 25, userAge[-2] = 24.

Data Types in Python VIII

- You can assign a list, or part of it, to a variable. If you write userAge2 = userAge, the variable userAge2 becomes [21, 22, 23, 24, 25].
- If you write userAge3 = userAge[2:4], you are assigning items with index 2 to index 4-1 from the list userAge to the list userAge3. In other words, userAge3 = [23, 24].
- The notation 2:4 is known as a slice. Whenever we use the slice notation in Python, the item at the start index is always included, but the item at the end is always excluded. Hence the notation 2:4 refers to items from index 2 to index 4-1 (i.e. index 3), which is why userAge3 = [23, 24] and not [23, 24, 25].
- The slice notation includes a third number known as the stepper. If we write userAge4 = userAge[1:5:2], we will get a sub list consisting of every second number from index 1 to index 5-1 because the stepper is 2. Hence, userAge4 = [22, 24].

UNIVERSITY OF GHANA

Data Types in Python IX

- In addition, slice notations have useful defaults. The default for the first number is zero, and the default for the second number is size of the list being sliced. For instance, userAge[:4] gives you values from index 0 to index 4-1 while userAge[1:] gives you values from index 1 to index 5-1 (since the size of userAge is 5, i.e. userAge has 5 items).
- To modify items in a list, we write listName[index of item to be modified] = new value. For instance, if you want to modify the second item, you write userAge[1] = 5. Your list becomes userAge = [21, 5, 23, 24, 25]
- To add items, you use the append() function. For instance, if you write userAge.append(99), you add the value 99 to the end of the list. Your list is now userAge = [21, 5, 23, 24, 25, 99]
- To remove items, you write del listName[index of item to be deleted]. For instance, if you write del userAge[2], your list now becomes userAge = [21, 5, 24, 25, 99] (the third item is deleted).

5. Tuple

Data Types in Python X

- Tuples are just like lists, but you cannot modify their values. The initial values are the values that will stay for the rest of the program. An example where tuples are useful is when your program needs to store the names of the months of the year.
- To declare a tuple, you write tupleName = (initial values). Notice that we use round brackets ( ) when declaring a tuple. Multiple values are separated by a comma.
  Example:

```
monthsOfYear = ("Jan", "Feb", "Mar", "Apr",
↪  "May", "Jun", "Jul", "Aug", "Sep", "Oct",
↪  "Nov", "Dec")
```

- You access the individual values of a tuple using their indexes, just like with a list. Hence, monthsOfYear[0] = “Jan”, monthsOfYear[-1] = “Dec”.

Data Types in Python XI

6. Dictionary
   - Dictionary is a collection of related data PAIRS. For instance, if we want to store the username and age of 5 users, we can store them in a dictionary.
   - To declare a dictionary, you write dictionaryName = dictionary key : data, with the requirement that dictionary keys must be unique (within one dictionary). That is, you cannot declare a dictionary like this

     ```
     myDictionary = {'Peter':38, 'John':51, 'Peter':13}
     ```

   - This is because "Peter" is used as the dictionary key twice. Note that we use curly brackets when declaring a dictionary. Multiple pairs are separated by a comma.
     Example:

     ```
     userNameAndAge = {'Peter':38, 'John':51, 'Alex':13,
     ↪  'Alvin':'Not Available'}
     ```

Data Types in Python XII

- You can also declare a dictionary using the dict( ) method. To declare the userNameAndAge dictionary above, you write

```
userNameAndAge = dict(Peter = 38, John = 51, Alex = 13,
↪  Alvin = 'Not Available')
```

- When you use this method to declare a dictionary, you use round brackets ( ) instead of curly brackets and you do not put quotation marks for the dictionary keys.
- To access individual items in the dictionary, we use the dictionary key, which is the first value in the dictionary key : data pair. For instance, to get John's age, you write

```
userNameAndAge['John']
```

  You'll get the value 51.
- To modify items in a dictionary, we write dictionaryName[dictionary key of item to be modified] = new data. For instance, to modify the "John":51 pair, we write

```
userNameAndAge['John'] = 21
```

Data Types in Python XIII

Our dictionary now becomes userNameAndAge ={“Peter”:38, “John”:21, “Alex”:13, “Alvin”:“Not Available”}

Making Choices & Decisions in Python

Condition Statements I

- All control flow tools involve evaluating a condition statement. The program will proceed differently depending on whether the condition is met.
- The most common condition statement is the comparison statement. If we want to compare whether two variables are the same, we use the == sign (double =). For instance, if you write x == y, you are asking the program to check if the value of x is equals to the value of y. If they are equal, the condition is met and the statement will evaluate to True. Else, the statement will evaluate to False.
- Other comparison signs include != (not equals), < (smaller than), > (greater than), <= (smaller than or equals to) and >= (greater than or equals to). The list below shows how these signs can be used and gives examples of statements that will evaluate to True.
  - Not equals: 5 != 2
  - Greater than: 5>2
  - Smaller than: 2<5

Condition Statements II

  - Greater than or equals to: 5>=2; 5>=5
  - Smaller than or equals to: 2 <= 5; 2 <= 2

- We also have three logical operators, and, or, not that are useful if we want to combine multiple conditions.
- The and operator returns True if all conditions are met. Else it will return False. For instance, the statement 5==5 and 2>1 will return True since both conditions are True.
- The or operator returns True if at least one condition is met. Else it will return False. The statement 5 > 2 or 7 > 10 or 3 == 2 will return True since the first condition 5>2 is True.
- The not operator returns True if the condition after the not keyword is false. Else it will return False. The statement not 2>5 will return True since 2 is not greater than 5.

Condition Statements III

- If Statement
  - The if statement is one of the most commonly used control flow statements. It allows the program to evaluate if a certain condition is met, and to perform the appropriate action based on the result of the evaluation. The structure of an if statement is as follows:

```
1  if condition 1 is met:
2    do A
3  elif condition 2 is met:
4    do B
5  elif condition 3 is met:
6    do C
7  elif condition 4 is met:
8    do D
9  else:
10   do E
```

Condition Statements IV

- elif stands for “else if” and you can have as many elif statements as you like.
- If you’ve coded in other languages like C or Java before, you may be surprised to notice that no parentheses ( ) are needed in Python after the if, elif and else keyword. In addition, Python **does not** use curly { } brackets to define the start and end of the if statement. Rather, Python uses indentation. Anything indented is treated as a block of code that will be executed if the condition evaluates to true.
- The program below first prompts the user for an input using the input function. The result is stored in the userInput variable as a string.

UNIVERSITY OF GHANA

Condition Statements V

```
1  userInput = input('Enter 1 or 2: ')
2
3  if userInput == "1":
4    print ("Hello World")
5    print ('How are you?')
6  elif userInput == "2":
7    print ('Python Rocks!')
8    print ('I love Python')
9  else:
10   print ('You did not enter a valid number')
```

- Inline If
  - An inline if statement is a simpler form of an if statement and is more convenient if you only need to perform a simple task. The syntax is:

    `do Task A if condition is true else do Task B`
  - For instance,

    ```
    num1 = 12 if myInt==10 else 13
    ```

Condition Statements VI

- This statement assigns 12 to num1 (Task A) if myInt equals to 10. Else it assigns 13 to num1 (Task B).
- Another example is

```
print ('This is task A' if myInt == 10 else 'This is
↪  task B')
```

- This statement prints "This is task A" (Task A) if myInt equals to 10. Else it prints "This is task B" (Task B).

- For Loop
  - Next, let us look at the for loop. The for loop executes a block of code repeatedly until the condition in the for statement is no longer valid.
  - Looping through an iterable: In Python, an iterable refers to anything that can be looped over, such as a string, list or tuple. The syntax for looping through an iterable is as follows:

```
1 for a in iterable:
2   print(a)
```

Condition Statements VII

- Example:

```
pets = ['cats', 'dogs', 'rabbits', 'hamsters']

for myPets in pets:
  print (myPets)
```

- Looping through a sequence of numbers: To loop through a sequence of numbers, the built-in range() function comes in handy. The range() function generates a list of numbers and has the syntax range (start, end, step).
- If start is not given, the numbers generated will start from zero.
- To see how the range() function works in a for statement, try running the following code:

```
  for i in range(5):
    print(i)
```

- While Loop

Condition Statements VIII

- The next control flow statement we are going to look at is the while loop. Like the name suggests, a while loop repeatedly executes instructions inside the loop while a certain condition remains valid. The structure of a while statement is as follows:

```
while condition is true:
  do A
```

- Most of the time when using a while loop, we need to first declare a variable to function as a loop counter. Let's just call this variable counter. The condition in the while statement will evaluate the value of counter to determine if it smaller (or greater) than a certain value. If it is, the loop will be executed. Let's look at a sample program.

```
counter = 5

while counter > 0:
  print ("Counter = ", counter)
  counter = counter - 1
```

Condition Statements IX

- At first look, a while statement seems to have the simplest syntax and should be the easiest to use. However, one has to be careful when using while loops due to the danger of infinite loops. Notice that in the program above, we have the line counter = counter - 1? This line is crucial. It decreases the value of counter by 1 and assigns this new value back to counter, overwriting the original value.
- We need to decrease the value of counter by 1 so that the loop condition while counter > 0 will eventually evaluate to False. If we forget to do that, the loop will keep running endlessly resulting in an infinite loop. If you want to experience this first hand, just delete the line counter = counter - 1 and try running the program again.

- Break: When working with loops, sometimes you may want to exit the entire loop when a certain condition is met. To do that, we use the break keyword. Run the following program to see how it works.

Condition Statements X

```
1  j = 0
2  for i in range(5):
3      j = j + 2
4      print ('i = ', i, ', j = ', j)
5      if j == 6:
6          break
```

Condition Statements XI

- Continue: Another useful keyword for loops is the continue keyword. When we use continue, the rest of the loop after the keyword is skipped for that iteration. An example will make it clearer.

```
1  j = 0
2  for i in range(5):
3    j = j + 2
4    print ('\n i = ', i, ', j = ', j)
5    if j == 6:
6      continue
7    print ('I will be skipped over if j=6')
```

- Try, Except:: The final control statement we'll look at is the try, except statement. This statement controls how the program proceeds when an error occurs. The syntax is as follows:

Condition Statements XII

```
try:
    do something
except:
    do something else when an error occurs
```

For instance, try running the program below

```
try:
    answer =12/0
    print (answer)
except:
    print("An error occurred")
```

la fin! la fin! la fin!

UNIVERSITY OF GHANA