Module 2 : Basic Applications of Programming in IoT

Presenter: Assoc. Prof. Ts Dr. Ahmad Shukri Mohd Noor

MODULE OUTLINE

2.1

MODULE 2.1

Introduction to Programming Languages

MODULE 2.2

Introduction to Python

2.2

2.3

MODULE 2.3

Variables and Data Types in Programming

MODULE 2.4

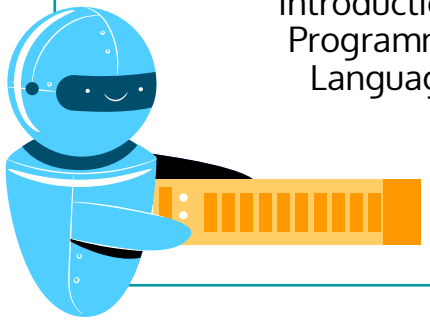
Basic Operators in Programming

2.4

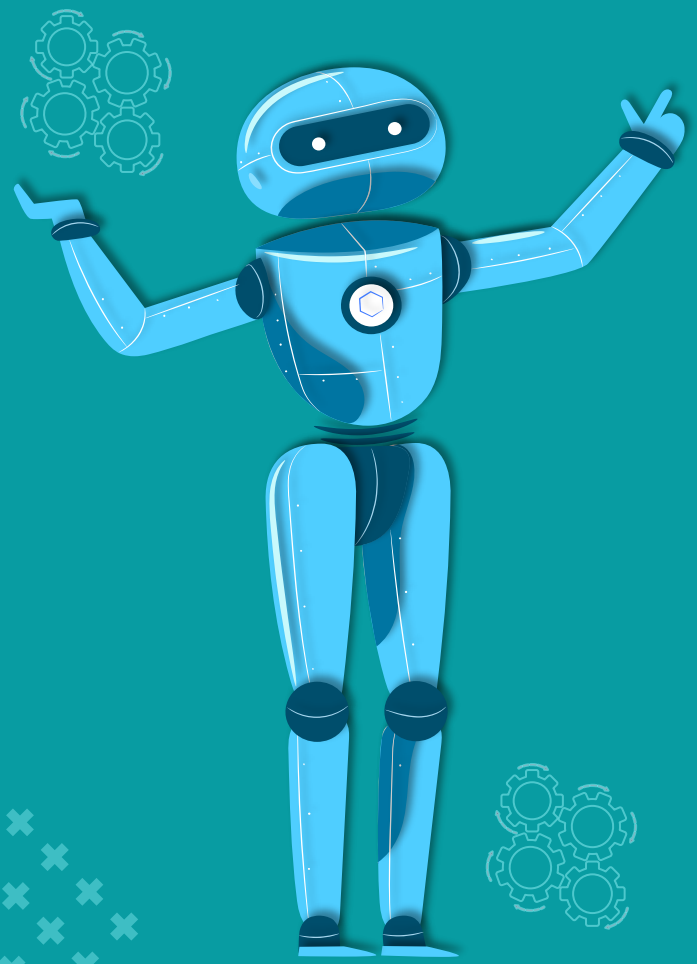
2.5

MODULE 2.5

Control Structures in Programming



2.2



Introduction to Python

- Let's Python!
- Python Versions
- Python Development Environments
- Python Elements
- Python Comments

2.2 Introduction to Python

2.2.1 Let's Python!

Python

- Python is a general purpose scripting language that implements the imperative, object-oriented, and functional paradigms.
- Dynamic typing, automatic memory management, exceptions, large standard library, modular.
- Extensions can be written in C and C++
- Other language versions (Jython, IronPython) support extensions written in Java and .Net languages)
- Design philosophy: easy to read, easy to learn

[redacted]

2.2 Introduction to Python

2.2.1 Let's Python!

Why python?

- The world's fastest growing programing language
- Among software engineer, mathematicians, data analyst, scientist, accountants, network engineer and even KIDS.
- A multi-purpose language with a simple and beginner-friendly syntax.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.
- It is a high level language
- Cross-platform
- Huge community
- Large ecosystem

[redacted]

2.2 Introduction to Python

2.2.1 Let's Python!

What can python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

2.2 Introduction to Python

2.2.2 Python Versions

There are **TWO** versions right now

- Python 2: compatible with much existing software
- Python 3: a major redesign
  - Not backward compatible.
  - Most features are the same or similar, but a few will cause older programs to break.
  - Part of the Python philosophy –don't clutter up the language with outdated features

Python Releases for Windows

- Latest Python 3 Release - Python 3.8.2
- Latest Python 2 Release - Python 2.7.17

[redacted]

2.2 Introduction to Python

2.2.3 Python Development Environments

- PyDev with Eclipse
- PyCharm
- Komodo
- Emacs
- Vim
- TextMate
- Gedit
- Idle
- PIDA (Linux)(VIM Based)
- NotePad++ (Windows)
- BlueFish (Linux)

[redacted]

2.2 Introduction to Python

Lab Guide

1. Install python for windows

➔ https://docs.google.com/document/d/1Kr5uXCNE9vxY8bqJ1uZPwPPW9Ja1oiFMMZ_VNxG1zJA/edit

1. Use jupyter notebook to code python

➔ https://docs.google.com/document/d/1nGRgdZyAkEJMckepJANgArJqzCrtbEUIVWHBtoVWdoM/edit

1. Use uPyCraft ide for micropython

➔ https://docs.google.com/document/d/1Kr5uXCNE9vxY8bqJ1uZPwPPW9Ja1oiFMMZ_VNxG1zJA/edit

[redacted]

2.2 Introduction to Python

2.2.4 Python Elements

- Syntax
  - Python syntax can be executed by writing directly in the Command Line
- Identifiers
  - Must begin with letter or underscore, followed by any number of letters, digits, underscores

```
In [1]: print("Hello world")

        Hello world
```

https://www.freecodecamp.org/news/a-gentler-introduction-to-programming-1f57383a1b2c/

[redacted]

2.2 Introduction to Python

2.2.4 Python Elements

- Python Indentation
  - Indentation refers to the spaces at the start of a line of code.
  - Whereas indentation in code is for readability only in other programming languages, the indentation in Python is very significant.
  - Python uses the indent to indicate a block of code.

```
In [2]: if 5>3:
        print("wrong indentation")

  File "<ipython-input-2-ed14205c5890>", line 2
    print("wrong indentation")
        ^
IndentationError: expected an indented block
```

```
In [3]: if 5>3:
            print("right indentation")

right indentation
```

2.2 Introduction to Python

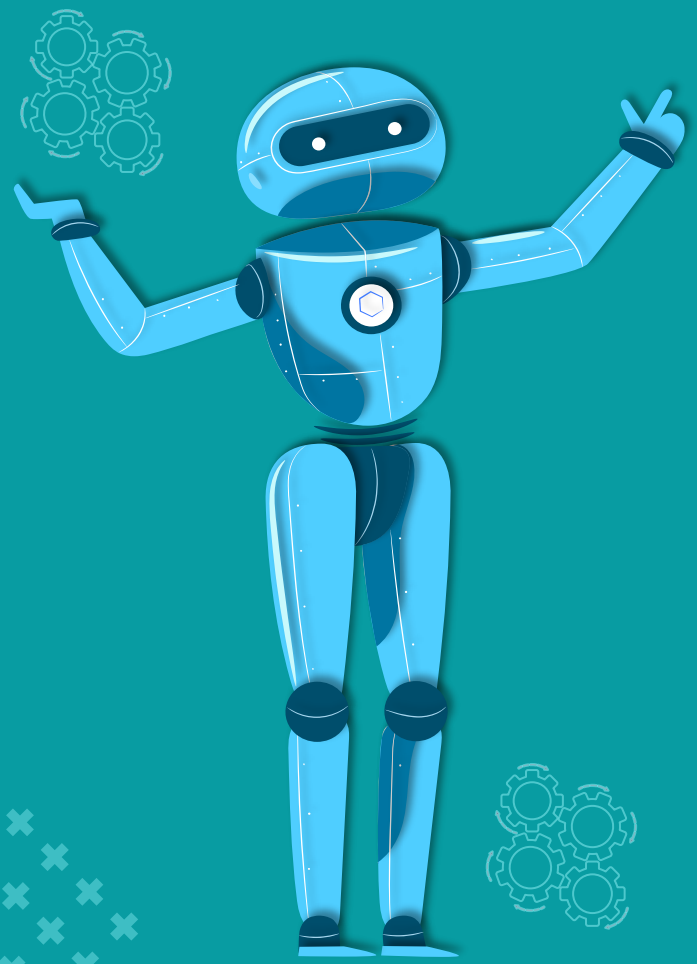
2.2.5 Python Comments

- Comments can be used to describe code in Python.
- Comments may be used to make the code readable.
- Comments may be used while checking code, to avoid execution.
- Comments start with a #, and will be ignored by Python.
- No, Python has no syntax for multi-line comments.

```
In [9]: # this is comment
        print("comment test")

        comment test
```

```
In [8]: # this is comment
        # python do not have multiline comment
        print("comment test")

        comment test
```

2.3



Variables and data types in programming

- Python Variables
- Python Built-in Data Types
- Python Input, Output and Import

2.3 Variables and data types in programming

2.3.1 Python Variables

- Variables are the containers where data values are stored.
- A variable is created the moment of first assign value to it.
- A name of a variable has to start with a letter or an underscore.
- A name variable can contain only alpha-numeric characters and underscores (A-z, 0-9, and).
- The names of the variables are case sensitive.

```
In [11]: a = 5
         b = 6.0098
         _car = 'BMW' # can use '' or " "

         print(_car)
         print(b)
         print(a)
```

```
BMW
6.0098
5
```

[redacted]

2.3 Variables and data types in programming

2.3.1 Python Variables

- Python allow us to assign value to multiple variables in single line.

```
In [13]: #value assigned to multiple variables

         car1,car2,car3 = "BMW",'Mercedes',"Volkswagen"
         print(car1)
         print(car2)
         print(car3)

         BMW
         Mercedes
         Volkswagen
```

- Python helps one to assign several variables in single line with the same value.

```
In [16]: #same value assigned to multiple variables

         car1=car2=car3 = "Viva"
         print(car1)
         print(car2)
         print(car3)

         Viva
         Viva
         Viva
```

[redacted]

2.3 Variables and data types in programming

2.3.1 Python Variables

- Often, the Python print statement is used to display variables.
- Python uses the + character to combine both text and a variable.
- The + character functions as a mathematical operator for numbers.
- Python will send you an error if you try to combine a string and a number.

```
In [18]: nama = 'Dan'
         print('My name is ' + nama)

         My name is Dan
```

```
In [19]: nama = 'Dan'
         ayat = 'My name is '
         print( ayat + nama)

         My name is Dan
```

```
In [20]: number1 = 20
         number2 = 2020
         print(number1 + number2)

         2040
```

[redacted]

2.3 Variables and data types in programming

2.3.1 Python Variables

- Variables created outside of a function are called global variables.
- Everybody may use global variables, both within and outside of the functions.
- If there is the same variable name as the global and local variable, the local variable may only be used within the function while the original global variable remains

```
In [22]: location = 'Sungai Petani'

         def function1():
             print(location)

         function1()
```

```
Sungai Petani
```

```
In [23]: #global variable
         location = 'Sungai Petani'

         def function1():
             #local variable
             location = 'Jitra'
             print(location)

         function1()
         print(location)
```

```
Jitra
Sungai Petani
```

[redacted]

2.3 Variables and data types in programming

2.3.1 Python Variables

- You may use the Global Keyword to create a global variable within a function.
- Also, if you want to modify a global variable within a function, use the global keyword

In [25]:
```
#global variable
location = 'Sungai Petani'

def function1():
    #global variable created
    #inside function
    global location

    location = 'Jitra'
    print(location)

function1()
print(location)
```

```
Jitra
Jitra
```

[redacted]

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Data type is an important concept in programming.
- Variables may store different types of data, and different types of data can do different things.
- Python has built-in data forms in these categories, by default:

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`

[redacted]

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Lists and tuples store, in a particular order, one or more items or values.
- The objects stored in a list or tuple may be of any form including the None Keyword specified form of nothing.
- In most cases the lists and tuples are identical but there are some differences

```
In [39]: e = ["wij","dan","mohamad"]
         print(e)
         print(type(e))

         ['wij', 'dan', 'mohamad']
         <class 'list'>

In [40]: f = ("wij","dan","mohamad")
         print(f)
         print(type(f))

         ('wij', 'dan', 'mohamad')
         <class 'tuple'>
```

[redacted]

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- The literal syntax of tuples is shown by parentheses () whereas the literal syntax of lists is shown by square brackets [] .
- Lists has variable length, tuple has fixed length.
- List has mutable nature, tuple has immutable nature.
- List has more functionality than the tuple.

```
In [41]: e = ["wij","dan","mohamad"]
         print(e)

         e[2] = "ariff"
         print(e)

         ['wij', 'dan', 'mohamad']
         ['wij', 'dan', 'ariff']
```

```
In [42]: f = ("wij","dan","mohamad")
         print(f)
         f[2] = ariff
         print(f)

         ('wij', 'dan', 'mohamad')

         ---------------------------------------------------------------------------
         NameError                                 Traceback (most recent call last)
         <ipython-input-42-07d2fee7e8f1> in <module>
               1 f = ("wij","dan","mohamad")
               2 print(f)
         ----> 3 f[2] = ariff
               4 print(f)

         NameError: name 'ariff' is not defined
```

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Lists has more builtin function than that of tuple. We can use dir([object]) inbuilt function to get all the associated functions for list and tuple.

```
In [43]: dir(e)
Out[43]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__delitem__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__gt__',
          '__hash__',
          '__iadd__',
          '__imul__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__reversed__',
          '__rmul__',
          '__setattr__',
          '__setitem__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          'append',
          'clear',
          'copy',
```

```
In [44]: dir(f)
Out[44]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__getnewargs__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__rmul__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          'count',
          'index']
```

[redacted]

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Tuples operation has smaller size than that of list, which makes it a bit faster.

```
In [45]: e = ["wij","dan","mohamad"]
         f = ("wij","dan","mohamad")

         print(e.__sizeof__())
         print(f.__sizeof__())

64
48
```

[redacted]

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Dictionary is an unordered collection of key-value pairs.
- It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
- Each item being a pair in the form key:value.
- Key and value can be of any type.

```
In [51]: h = {'name': 'wijdan', 'age': 20}

         print("his name is",h['name'])

         print("his age is",h['age'])
```

```
his name is wijdan
his age is 20
```

[redacted]

2.3 Variables and data types in programming

2.3.2 Python Built-in Data Types

- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.
- We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.
- Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

```
In [53]: i = {'dan', 'dan', 'dan', 'wij', 'mohamad'}
         print(i)
         print(i[1])

         {'dan', 'wij', 'mohamad'}

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-53-1bf0bce0a741> in <module>
               1 i = {'dan', 'dan', 'dan', 'wij', 'mohamad'}
               2 print(i)
         ----> 3 print(i[1])

         TypeError: 'set' object does not support indexing
```

2.3 Variables and data types in programming

2.3.3 Python Input, Output and Import

- We use the print() function to output data to the standard output device (screen).
- We can also output data to a file.
- The actual syntax of the print() function is

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
In [ ]: print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

value to be printed

separator used between values

printed after all values are printed. default is new line

sys.stdout is screen as default

```
In [1]: print(1,2,3,4)
        print(1,2,3,4, sep='#', end='.')

1 2 3 4
1#2#3#4.
```

[redacted]

2.3 Variables and data types in programming

2.3.3 Python Input, Output and Import

- Sometimes we would like to format our output to make it look attractive.
- This can be done by using the str.format() method. This method is visible to any string object.

```
In [2]: x = 10
        y = 2020

        print("I am {} years old in {}".format(x,y))
```

```
I am 10 years old in 2020
```

```
In [3]: print("i love {0} and {1}".format("roti canai","teh tarik"))
        print("i love {1} and {0}".format("roti canai","teh tarik"))
```

```
i love roti canai and teh tarik
i love teh tarik and roti canai
```

[redacted]

2.3 Variables and data types in programming

2.3.3 Python Input, Output and Import

- To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

```
input([prompt])
```

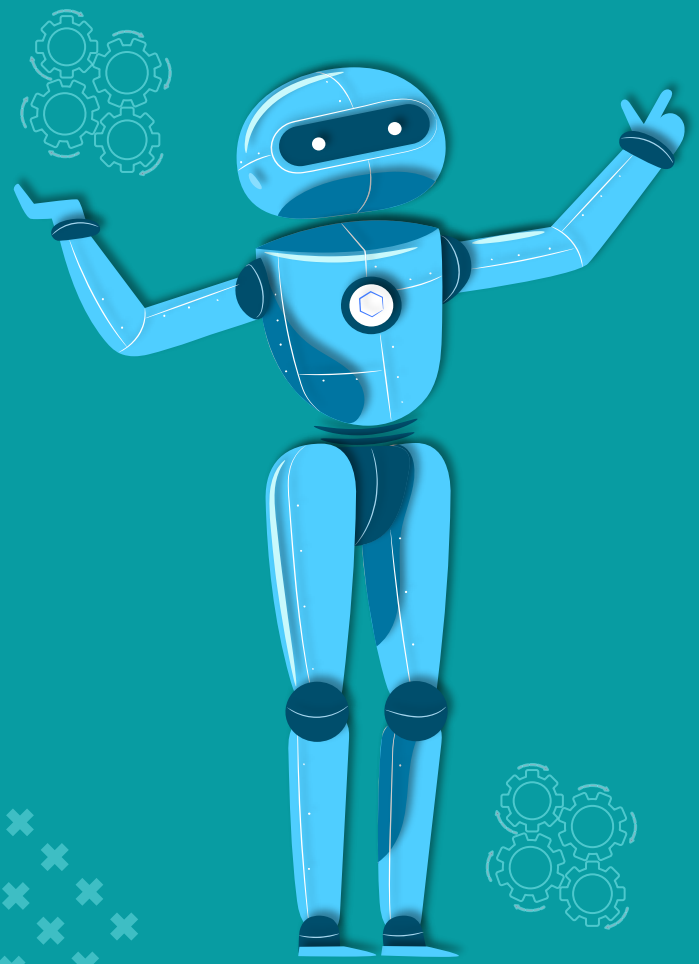
- It is save in string data type
- Use a cast to take numeric data

```
In [5]: z = input('Enter a number :')
        z

        Enter a number :200

Out[5]: '200'
```

```
In [20]: name,age = input('enter your name:'),int(input('enter your age:'))

         enter your name:wijdan
         enter your age:20
```

[redacted]

2.4



Basic operators in programming

- Python Operators - Arithmetic
- Python Operators - Comparison
- Python Operators - Logical
- Python Operators - Bitwise
- Python Operators - Assignment
- Python Namespace

2.4 Basic Operators in Programming

2.4.1 Python Operators - Arithmetic

- Assume variable x holds 2 and variable y holds 4, then –

```
In [8]: x = 2
        y = 4
        print("x + y = ", x+y)
        print("x - y = ", x-y)
        print("x * y = ", x*y)
        print("x / y = ", x/y)
        print("x // y = ", x//y)
        print("x ** y = ", x**y)
```

```
x + y =  6
x - y =  -2
x * y =  8
x / y =  0.5
x // y =  0
x ** y =  16
```

2.4 Basic Operators in Programming

2.4.2 Python Operators - Comparison

- These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.
- Assume variable x holds 2 and variable y holds 4, then –

```
In [10]: x = 2
         y = 4
         print("x > y = ", x>y)
         print("x < y = ", x<y)
         print("x == y = ", x==y)
         print("x != y = ", x!=y)
         print("x >= y = ", x>=y)
         print("x <= y = ", x<=y)
```

```
x > y =  False
x < y =  True
x == y =  False
x != y =  True
x >= y =  False
x <= y =  True
```

2.4 Basic Operators in Programming

2.4.3 Python Operators - Logical

- There are following logical operators supported by Python language.
- Assume variable x holds 2 and variable y holds 4, then –

```
In [11]: x = True
         y = False
         print("x and y = ", x>y)
         print("x or y = ", x<y)
         print("x not y = ", x==y)
```

```
x and y =  True
x or y =  False
x not y =  False
```

[redacted]

2.4 Basic Operators in Programming

2.4.4 Python Operators - Bitwise

- Bitwise operator works on bits and performs bit by bit operation.
- Assume if x = 8; and y = 4; Now in the binary format their values will be 1000 and 0100 respectively.

```
In [13]: x = 8
         y = 4

         print(x&y) #and
         print(x|y) #or
         print(~x) #not
         print(x^y) #exclusive or
         print(x>>2) #bitwise right shift
         print(x<<2) #bitwise left shift
```

```
0
12
-9
12
2
32
```

2.4 Basic Operators in Programming

2.4.5 Python Operators - Assignment

- Assigns values from right side operands to left side operand

Assignment operators in Python		
Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

[redacted]

2.4 Basic Operators in Programming

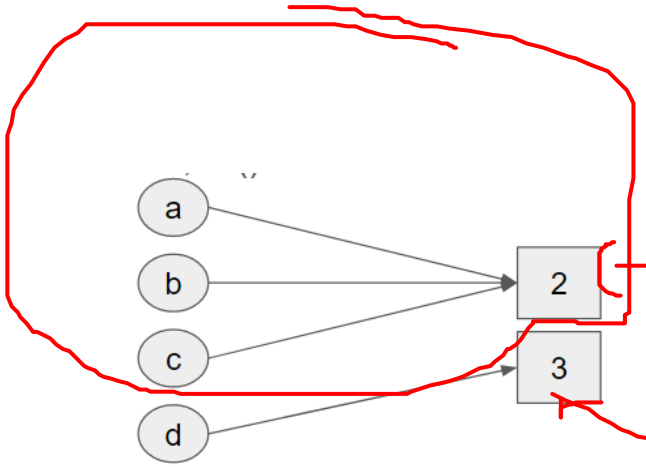
2.4.6 Python Namespace

- Name (also called identifier) is simply a name given to objects.
- Everything in Python is an object.
- Name is a way to access the underlying object.
- We can get the address (in RAM) of some object through the built-in function, id()

```
In [14]: a = 2
         print("address for name 2", id(a))
         print("address for object a", id(2))
```

```
address for name 2 140704690135920
address for object a 140704690135920
```

```
In [17]: a = 2
         b = a
         c = 2
         d = 3
         print("address for a", id(a))
         print("address for b", id(b))
         print("address for c", id(c))
         print("address for d", id(d))
```



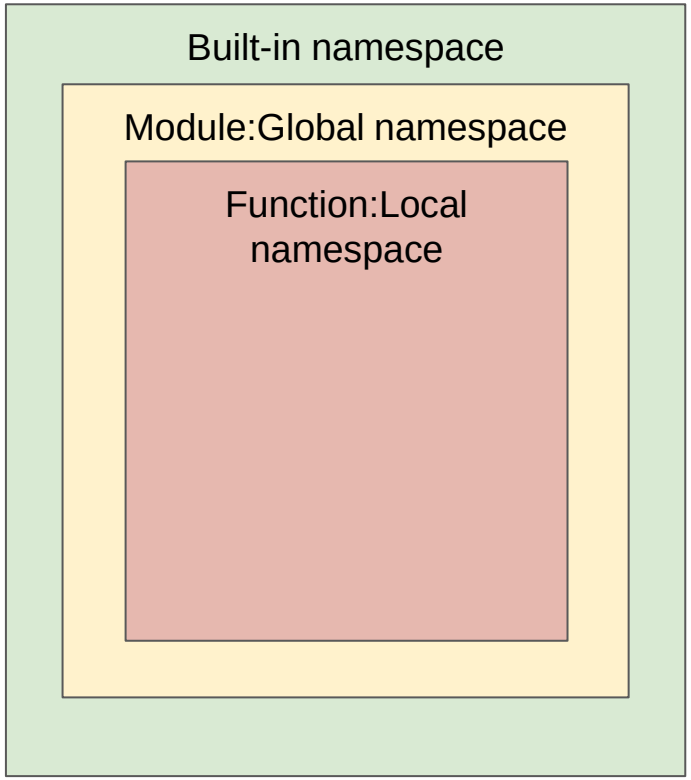

```
address for a 140704690135920
address for b 140704690135920
address for c 140704690135920
address for d 140704690135952
```

[redacted]

2.4 Basic Operators in Programming

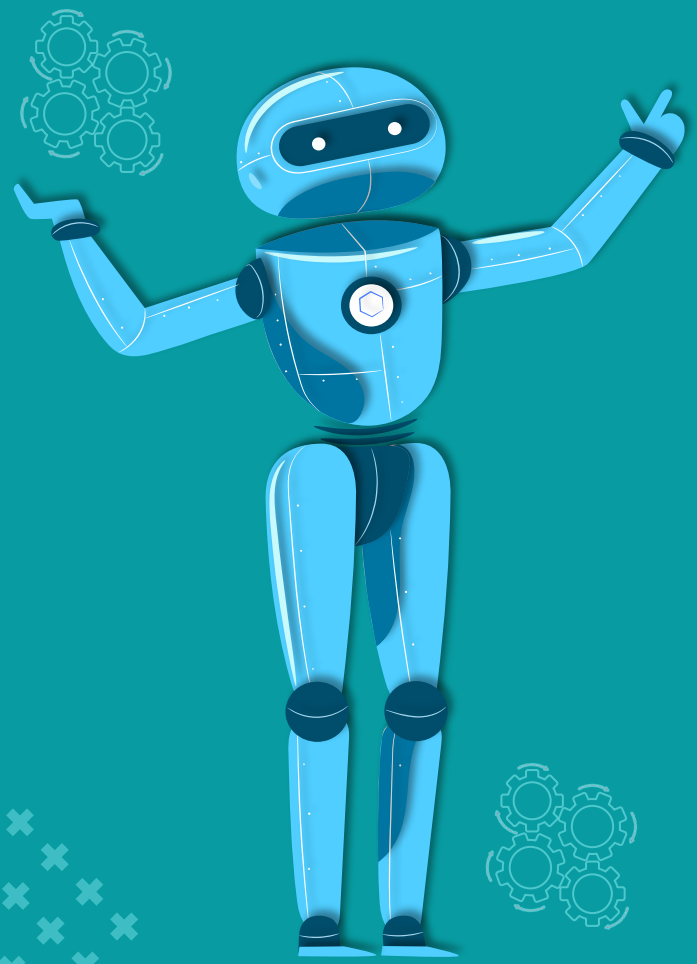
2.4.6 Python Namespace

- Namespace is collection of name
- Built-in names is created when Python started.
- Each module creates its own global namespace
- Local namespace is created when a function is called



[redacted]

2.5



Control Structures in Programming

- Python Control Structure - If..Else
- Python Control Structure - For loop
- Python Control Structure - while loop
- Python Function
- Python Exercise

2.5 Control Structures in Programming

2.5.1 Python Control Structure - If..Else

- Decision-making is needed when we only want a code to be executed if a certain requirement is met.
- The program evaluates the condition and will execute statements if the condition result is True
- Python take non-zero values as True, None and 0 as False.

```
if condition:
        statement(s)
elif condition:
        statement(s)
else:
        statement(s)
```

```
In [21]: value = int(input('enter a number:'))

         if value > 0:
             print('positive number')
         elif value == 0:
             print('zero')
         else:
             print('negative number')
```

```
enter a number:20
positive number
```

2.5 Control Structures in Programming

2.5.2 Python Control Structure - For loop

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- Here, val is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.

```
for val in sequence:
        Body of for
```

```
In [22]: car = ['BMW','Merc','Proton']
         for x in car:
             print(x)

BMW
Merc
Proton
```

```
In [24]: for x in 'Mercedes':
             print(x)

M
e
r
c
e
d
e
s
```

[redacted]

2.5 Control Structures in Programming

2.5.3 Python Control Structure - while loop

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is True.
- We generally use this loop when we don't know beforehand, the number of times to iterate.

```
while condition:
        Body of while
```

In [39]:
```
a = 1
b = 10

while a < b:
    print('a lower than b')
    a = a+1
```

```
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
```

2.5 Control Structures in Programming

2.5.4 Python Function

- In Python, function is a collection of associated statements that perform a specific task.
- Functions help break into smaller and more flexible parts of our program. As our system grows bigger and bigger, it's more structured and manageable by functions.
- It also prevents repetition, and makes code reusable.

```
def function_name(parameters):
        """This function……"""
    statement(s)
```

```
In [43]: def my_function():
             """This function to
             print hello"""

             print('Hello')

         my_function()
```

```
Hello
```

```
In [42]: def my_function():
             """This function to make
             addition between a and b"""
             a = int(input('a:'))
             b = int(input('b:'))
             print(a+b)

         my_function()
```

```
a:20
b:30
50
```

[redacted]

2.5 Control Structures in Programming

2.5.5 Python Exercise

- Create a function to determine fever
- When the function is called
  - Ask to enter body temperature
  - Answer whether or not you have a fever
    - 38 and above - fever
    - Below than 38 - healthy

```
Enter your body temperature: 38
Enter your body temperature:38
You have a fever. Go to the clinic.
```

```
Enter your body temperature: 37
Enter your body temperature:37
You are healthy.
```

[redacted]