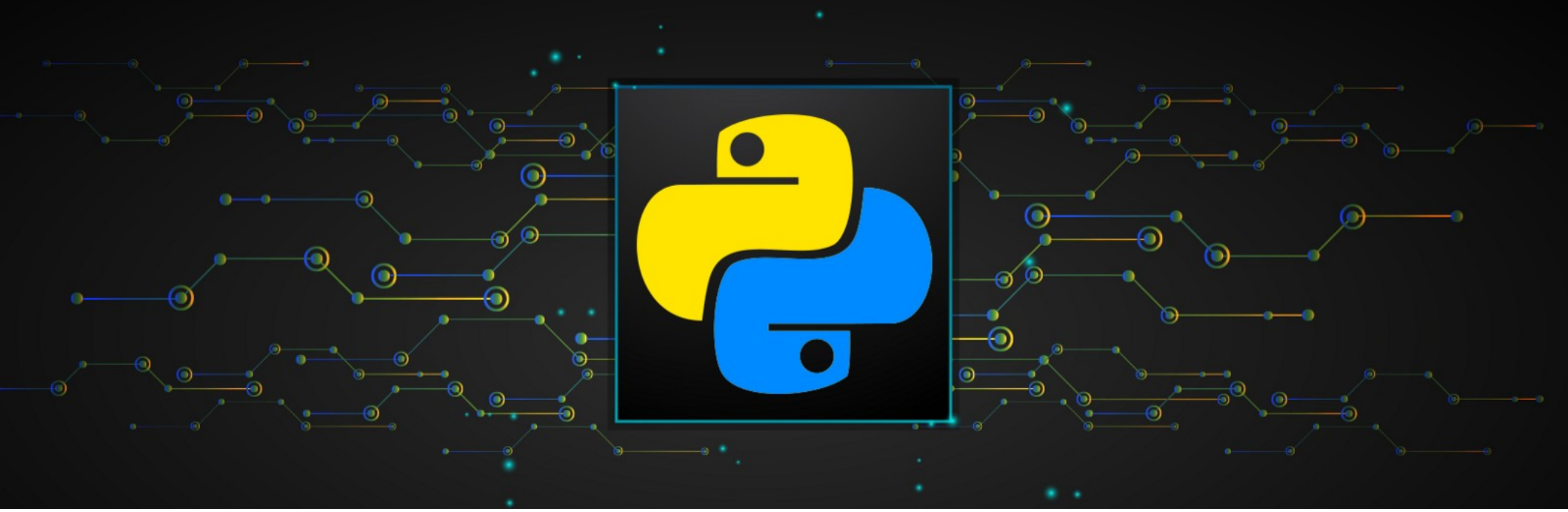


Python 3 Programming language

Start the future with it

introduction

- Python is an interpreted high-level programming language for general-purpose programming.

- Created by Guido van Rossum and first released in 1991.

Python's features

- Uses an elegant syntax, making the programs you write easier to read.
- Comes with a large standard library that supports many common task such as web development (server-side), software development, mathematics, map algebra.
- Can also be embedded into an application to provide a programmable interface.
- Runs anywhere, including Mac OS X, Windows, Linux, and Unix, also available for Android and iOS.
- Is free software in two senses. It doesn't cost anything to download or use Python.

- A variety of basic data types are available: numbers, lists, and dictionaries.
- Python supports object-oriented programming.
- Code can be grouped into modules and packages.
- The language supports exceptions, resulting in cleaner error handling.
- Data types are strongly and dynamically typed. Mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised, so errors are caught sooner.
- Python's automatic memory management frees you from having to manually allocate and free memory in your code.

Python Variables

Creating Variables

Unlike other programming languages, Python has no command for declaring a variable.

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

Example

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

Remember that variables are case-sensitive

Output Variables

- The Python print statement is often used to output variables.
- To combine both text and a variable, Python uses the + character:

Example
```
x = "awesome"
print("Python is " + x)
```

RUN

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

Not RUN

Python Numbers

- There are three numeric types in Python:
  - int
  - float
  - complex
- Variables of numeric types are created when you assign a value to them:

Example

```
x = 1    # int is a whole number, positive or negative, without decimals, of unlimited length.

y = 2.8  # float is a number, positive or negative, containing one or more decimals, can also be scientific numbers with an "e" to indicate t the power of 10.

z=35e3
```

Python Casting

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number).
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer).
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals.

Example

Integers:

```
x = int(1)   # x will be 1
y = int(3.8) # y will be 3
z = int("4") # z will be 4
```

Example

Floats:

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Example

Strings:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

When we want binary system number we must casting between integer and biary number system

In Python, you can simply use the bin() function to convert from a decimal value to its corresponding binary value.

And similarly, the int() function to convert a binary to its decimal value. The int() function takes as second argument the base of the number to be converted, which is 2 in case of binary numbers.

```
a = 79
# Base 2(binary)
bin_a = bin(a)
print(bin_a)
print(int(bin_a, 2))
#Base 2(binary)
```

Python Strings

- Python are arrays of bytes representing Unicode characters.
- Python does not have a character data type, a single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.

Example

```
a = "Hello, World!"

print(a[1])
print(a[2:5])
print(len(a))
```

```
print(a.lower())
print(a.upper())
print(a.replace("H", "J"))
print(a.split(",")) # returns ['Hello', '
World!']
```

Python Boolean

- The bool class is a subclass of int (It cannot be sub-classed further. Its only instances are False and True.

```
checker = None # not necessary

if some_decision:
    checker = True
if checker:
    # some stuff
```

Python Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise 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

Logical operators are used to combine conditional statements:

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)

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Data Structure

Muamar Almani Jasim
Technical Collage University
Collage of Technical -Kirkuk
2018-2019

List

- A list is a collection which is ordered and changeable.
- In Python lists are written with square brackets.

- **Example** Create a List:

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

Access Items

- you access the list items by referring to the index number:

- **Example** Print the second item of the list:

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

Change Item Value

- to change the value of a specific item, refer to the index number:

- **Example** Change the second item:

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

Loop Through a List

- you can loop through the list items by using a for loop:

- **Example** Print all items in the list, one by one:

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

Check if Item Exists

- To determine if a specified item is present in a list use the in keyword:

- **Example** Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

List Length

- to determine how many items a list have, use the len() method:

- **Example** Print the number of items in the list:

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

Add Items

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

- **Example** Using the append() method to append an item:

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

- To add an item at the specified index, use the insert() method:

- **Example** Insert an item as the second position:

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

Remove Item

- There are several methods to remove items from a list:

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

```
thislist = ["apple", "banana", "cherry"]

thislist.remove("banana")

print(thislist)
```

- The pop() method removes the specified index, (or the last item if index is not specified):

- **Example** The pop() method removes the specified index, (or the last item if index is not specified):

```
thislist = ["apple", "banana", "cherry"]

thislist.pop()

print(thislist)
```

- The del keyword removes the specified index:

- **Example** The del keyword removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

- The del keyword can also delete the list completely:

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

- #this will cause an error because "thislist" no longer exists.

- The clear() method empties the list:

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

The list() Constructor

- It is also possible to use the list() constructor to make a list.

- Example Using the list() constructor to make a List:
- thislist = list(("apple", "banana", "cherry"))

  # note the double round-brackets
- print(thislist)

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

TUPLE

- One dimensional , fixed-length, **immutable** sequence of python objects of ANY type.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses.
- In Python tuples are written with round brackets. ( )

- **Example** Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets:

- **Example** Return the item in position 1:

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

Loop Through a Tuple

- you can loop through the tuple items by using a for loop.

- **Example:**

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Change Tuple Values / Add Items/ Remove Items

- Once a tuple is created, you cannot change its values, you cannot add new item, you cannot add new item. Tuples are unchangeable.

- **Example:**

```
thistuple = ("apple", "banana", "cherry")
thistuple[1] = "blackcurrant"
# ERROR:
print(thistuple)
```

Check if Item Exists

- To determine if a specified item is present in a tuple use the in keyword:

- **Example:** Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

The tuple() Constructor

- It is also possible to use the **tuple( )** constructor to make a tuple.

- **Example:**Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry"))
# note the double round-brackets
print(thistuple)
```

Tuple Methods

- Python has two built-in methods that you can use on tuples.
  - count() :Returns the number of times a specified value occurs in a tuple
  - index() :Searches the tuple for a specified value and returns the position of where it was found

Exercise:Print the first item in the fruits tuple.

fruits = ("apple", "banana", "cherry")

Print( )

SET

- A set is an unordered collection of UNIQUE elements.
- A set is a collection which is unindexed.
- In Python sets are written with curly brackets. { }

Example Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Note: Sets are unordered, so the items will appear in a random order.

Access Items / Change Items

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

- Once a set is created, you cannot change its items, but you can add new items.

Add Items

- To add one item to a set use the add() method.
- To add more than one item to a set use the update() method.

- **Example** Add multiple items to a set, using the update() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)

#note [ ] when add using update
```

Remove Item

- To remove an item in a set, use the remove(), or the discard() method.
- The deference is If the item to remove does not exist, remove() will raise an error while discard will not.
- **Example** :Remove "banana" by using the discard() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

- The clear() method empties the set.

```
Listname.clear()
```

- The del keyword will delete the set completely.

```
del listname
```

The set() Constructor

- It is also possible to use the set() constructor to make a set.
- **Example** Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry"))
# note the double round-brackets
print(thisset)
```

Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {"brand": "Ford",  "model": "Mustang",  "year": 1964}
print(thisdict)
```

Out put will be : {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Accessing Item

- A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.
- **Example: Get the value of the "model" key:**

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

Example:Get the value of the "model" key:

```
x = thisdict.get("model")
```

Change Values

- You can change the value of a specific item by referring to its key name.
- **Example** :Change the "year" to 2018:

```
thisdict = {"brand":"Ford","model":"Mustang","year":1964}

thisdict["year"] = 2018
```

Loop Through a Dictionary

- You can loop through a dictionary by using a for loop.
- When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.
- **Example**:Print all key names in the dictionary, one by one:

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

- **Example**:Print all values in the dictionary, one by one:

```
for x in thisdict:
print(thisdict[x])
```

- **Example**: You can also use the values() function to return values of a dictionary:

  for x in thisdict.values():

  print(x)

- **Example**:Loop through both keys and values, by using the items() function:

  for x, y in thisdict.items():

  print(x, y)

Check if Key Exists

- To determine if a specified key is present in a dictionary use the in keyword.

- **Example:** Check if "model" is present in the dictionary:

```
thisdict = {"brand":"Ford","model":"Mustang","year":1964}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Dictionary Length

- To determine how many items (key-value pairs) a dictionary have, use the len() method.
- **Example:** Print the number of items in the dictionary:

```
print(len(thisdict))
```

Adding Items

- Adding an item to the dictionary is done by using a new **index** key and assigning a value to it:
- **Example**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Removing Items

- There are several methods to remove items from a dictionary:
- The pop() method removes the item with the specified key name: **thisdict.pop("model")**
- The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

  thisdict.popitem()
- The del keyword removes the item with the specified key name:

  del thisdict["model"]
- The keyword del can also delete the dictionary completely:**del thisdict**
- The clear() keyword empties the dictionary: **thisdict.clear()**

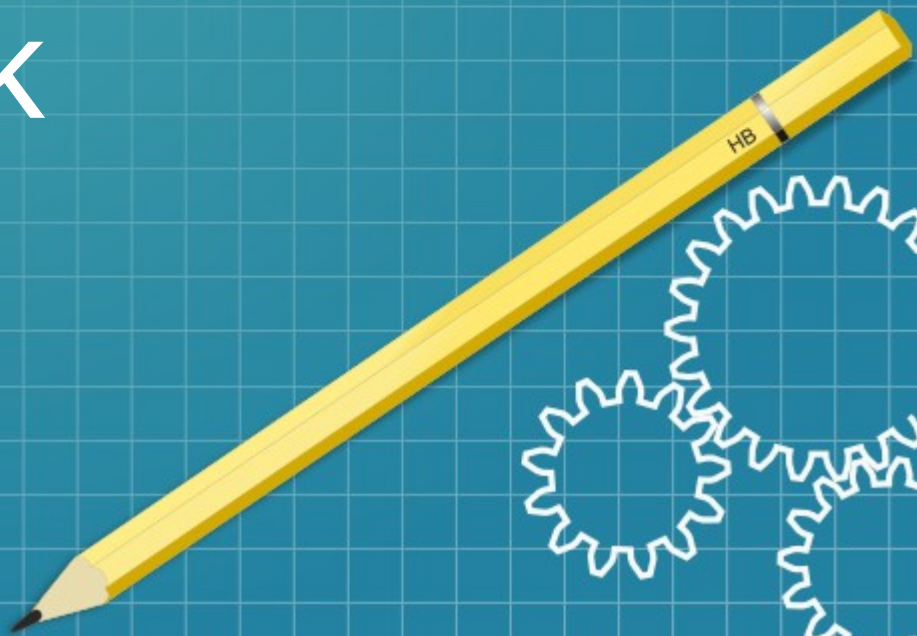
Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing the a tuple for each key value pair
keys()	Returns a list contianing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Data Structure in Python

Stack



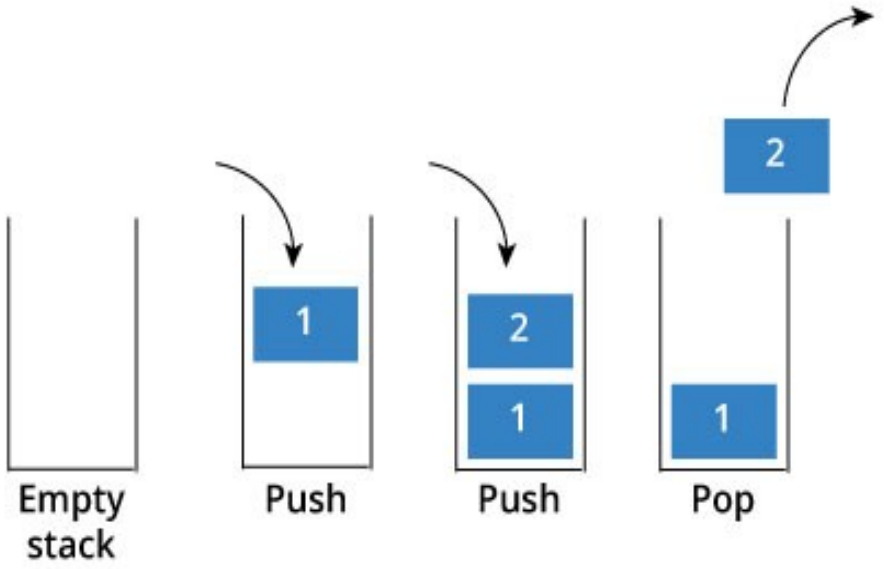



What is Stack ?

- A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.
- In the push down stacks only two operations are allowed:
  1. push the item into the stack.
  2. pop the item out of the stack.

- A stack is a limited access data structure - elements can be added and removed from the stack only at the top.

- push adds an item to the top of the stack.
- pop removes the item from the top.



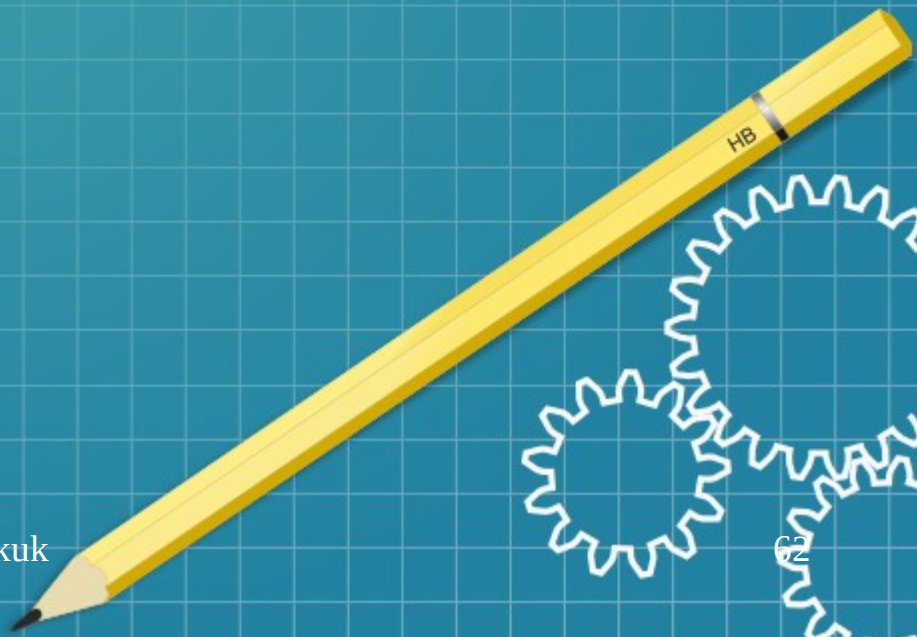
Stack Application

- Reversing a String.
- Parenthesis matching.
- Matching HTML Tags.
- Arithmetic Expression
  1. Infix to Postfix notation : (a+b)*c ab+c*
  2. Infix to prefix notation : (a+b)*c *+abc

```
class Stack:
    def __init__(self):
        self.stack = [ ]

    def push(self, data):
        self.stack.append(data)

    def pop(self):
        if self.stack:
            return self.stack.pop()
        else:
            print("Stack is empty")

    def __str__(self):
        return str(self.stack)

if __name__ == "__main__":
    stack = Stack()
    stack.push("A")
    stack.push("B")
    stack.push("C")
    print(stack)
    print("Pop :", stack.pop())
    print("current State :", stack)
```

output

```
['A', 'B', 'C']
Pop : C
current State : ['A', 'B']
```



Data Structure in Python

Queue

What is Queue ?

- A queue is a collection of objects that supports fast first-in, first-out (FIFO) semantics for inserts and deletes.
- The insert and delete operations sometimes called **enqueue** and **dequeue**.
- Unlike lists or arrays, queues typically don’t allow for random access to the objects they contain.

- Insertion performed at one end (Rear) .
- Deletion performed other end (Front) .

Queue Application

- Scheduling, CPU and Disk Scheduling.
- Parallel programming problems.
- A short and beautiful algorithm using a queue is breadth-first search (BFS) on a tree or graph data structure.
- Data transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

```
class Queue:

    def __init__(self):

        self.queue = []

    def enqueue(self, data):
        # Insert method to add element
        if data not in self.queue:
            self.queue.insert(0, data)
            return True
        else:
            return False

    # Pop method to remove element
    def dequeue(self):
        if len(self.queue) > 0:
            return self.queue.pop()
        else:
            return "No elements in Queue!"

    def isEmpty(self):
        return self.queue == []

    def __str__(self):
        return str(self.queue)

    def size(self):
        return len(self.queue)
```

```
class Queue:

    def __init__(self):

        self.queue = []

    def enqueue(self, data):
        # Insert method to add element
        if data not in self.queue:
            self.queue.insert(0, data)
            return True
        else:
            return False

    # Pop method to remove element
    def dequeue(self):
        if len(self.queue) > 0:
            return self.queue.pop()
        else:
            return "No elements in Queue!"

    def isEmpty(self):
        return self.queue == []

    def __str__(self):
        return str(self.queue)

    def size(self):
        return len(self.queue)
```