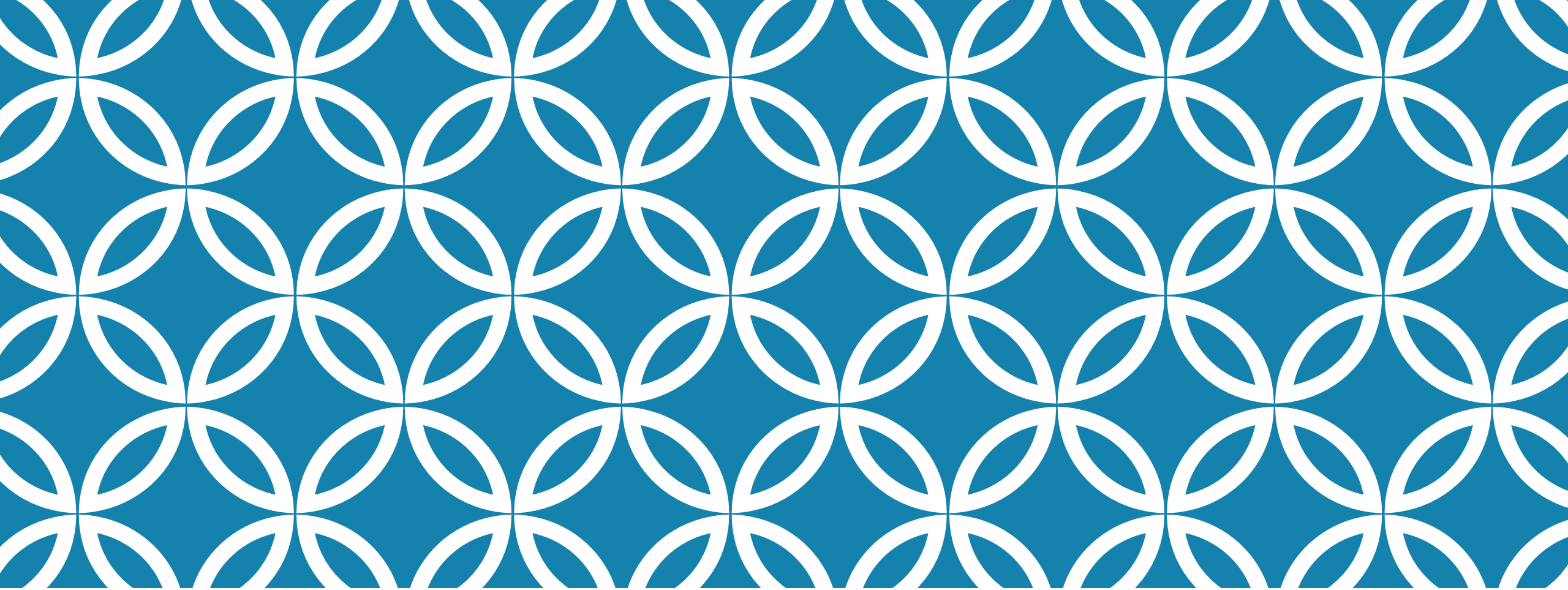


INTRODUCTION TO PYTHON

Lecture 3 (week-2)

OVERVIEW

Brief History of Python

Basic Datatypes

- Numbers
- Strings
- Lists
- Tuples
- Dictionaries

Variables

Control Structures

Functions

BRIEF HISTORY OF PYTHON

Invented in the Netherlands, early 90s by Guido van Rossum

Named after Monty Python

Open sourced from the beginning

Considered a **scripting** language, but is much more

Scalable, object-oriented and functional from the beginning

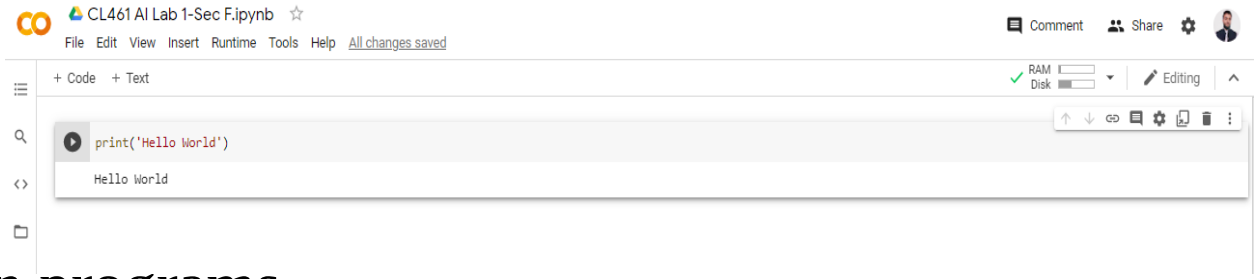
Used by Google from the beginning

Increasingly popular

"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

- Guido van Rossum

ONLINE PYTHON INTERPRETER



Working on Google Collab to run Python programs.

Instructions:

- Visit the following link: https://colab.research.google.com/
- Sign in using your nu email id
- You will be directed to 'Welcome to Colaboratory' page.
  - Go to **File->New Notebook**.
  - Write your first Python program by typing the following statement in the first cell
  - Print('Hello world')
  - Execute this cell by clicking the play button on the left of the cell or by pressing Ctrl+Enter.

BASIC DATATYPES

Numbers

String

List

Tuple

Dictionary

Sr#	Categories	Data Type	Examples
1	**Numeric Types**	int	-2, -1, 0, 1, 2, 3, 4, 5, int(20)
2		float	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25, float(20.5)
3		complex	1j, complex(1j)
4	**Text Sequence Type**	str	'a', 'Hello!', str("Hello World")
5	**Boolean Type**	bool	True, False, bool(5)
6	**Sequence Types**	list	["apple", "banana", "cherry"], list(("apple", "banana", "cherry"))
7		tuple	("apple", "banana", "cherry"), tuple(("apple", "banana", "cherry"))
8		range	range(6)
9	**Mapping Type**	dict	{"name" : "John", "age" : 36}, dict(name="John", age=36)
10	**Set Types**	set	{"apple", "banana", "cherry"}, set(("apple", "banana", "cherry"))
11		frozenset	frozenset({"apple", "banana", "cherry"})
12	**Binary Sequence Types**	bytes	b"Hello", bytes(5)
13		bytearray	bytearray(5)
14		memoryview	memoryview(bytes(5))

NUMBERS

Number data types store numeric values. Number objects are created when you assign a value to them. For example

```
var1 = 1
```

The usual suspects

- 12, 3.14, 0xFF, (-1+2)*3/4**5, abs(x), 0<x<=5

C-style shifting & masking

- 1<<16, x&0xff, x|1, ~x, x^y

OPERATORS

Math Operators

Operators	Operation	Example
**	Exponent	2 ** 3 = 8
%	Modulus/Remainder	22 % 8 = 6
//	Integer division	22 // 8 = 2
/	Division	22 / 8 = 2.75
*	Multiplication	3 * 3 = 9
-	Subtraction	5 - 2 = 3
+	Addition	2 + 2 = 4

Comparison Operators

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater Than
<=	Less than or Equal to
>=	Greater than or Equal to

Bitwise Operators

Operator	Example	Meaning
&	a & b	Bitwise AND
|	a | b	Bitwise OR
^	a ^ b	Bitwise XOR (exclusive OR)
~	~a	Bitwise NOT
<<	a << n	Bitwise left shift
>>	a >> n	Bitwise right shift

STRINGS

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search

LISTS

Represent ordered sequences of values of Mixed types.

Mutable

Flexible arrays.

Examples:

- items = [32, 100.25, “Apple”]
- a = [98, "bottles of beer", ["on", "the", "wall"]]
- del a[-1] # -> [98, "bottles", "of", "beer"]

MORE LIST OPERATIONS

```
>>> a = range(5)          # [0,1,2,3,4]
>>> a.append(5)           # [0,1,2,3,4,5]
>>> a.pop()                    # [0,1,2,3,4]

>>> a.insert(0, 42)            # [42,0,1,2,3,4]
>>> a.pop(0)                   # [0,1,2,3,4]

>>> a.reverse()           # [4,3,2,1,0]
>>> a.sort()              # [0,1,2,3,4]
```

TUPLES

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas.

Unlike lists, however, tuples are enclosed within parentheses.

Often used for functions that have multiple return values.

The **main differences** between lists and tuples are:

- Lists are enclosed in brackets ([ ]) and their elements and size can be changed,
- tuples are enclosed in parentheses (( )) and cannot be updated(immutable).
- Tuples can be thought of as read-only lists.

TUPLE (CONT'D)

Examples:

- t = (1, 2, 3)

Only 2 functions used

Count()

- thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
  x = thistuple.count(5)# 2

Index()

- x = thistuple.index(8) # 3

DICTIONARIES

Hash tables, "associative arrays"

- d = {"duck": "eend", "water": "water"}

Lookup:

- d["duck"] -> "eend"
- d["back"] # raises KeyError exception

Delete, insert, overwrite:

- del d["water"] # {"duck": "eend"}
- d["back"] = "rug" # {"duck": "eend", "back": "rug"}
- d["duck"] = "duik" # {"duck": "duik", "back": "rug"}

DICTIONARIES

Keys, values, items:

- d.keys() -> ["duck", "back"]
- d.values() -> ["duik", "rug"]
- d.items() -> [("duck","duik"), ("back","rug")]

Presence check:

- d.has_key("duck") -> 1; d.has_key("spam") -> 0

Values of any type; keys almost any

- {"name":"Guido", "age":43, ("hello","world"):1,
  42:"yes", "flag": ["red","white","blue"]}

DICTIONARIES VS LISTS

Dictionaries and lists **share** the following characteristics:

- Both are mutable.
- Both are dynamic. They can grow and shrink as needed.
- Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Dictionaries **differ** from lists primarily in how elements are accessed:

- List elements are accessed by their position in the list, via indexing.
- Dictionary elements are accessed via keys not by numerical index.

VARIABLE

No need to declare

Need to assign (initialize)

- use of uninitialized variable raises exception

Not typed

```
if friendly:
    greeting = "hello world"
else:
    greeting = 12
```

Everything is a "variable":

- Even functions, classes, modules

REFERENCE SEMANTICS

Assignment manipulates references

- x = y **does not make a copy** of y
- x = y makes x **reference** the object y references

Very useful; but beware!

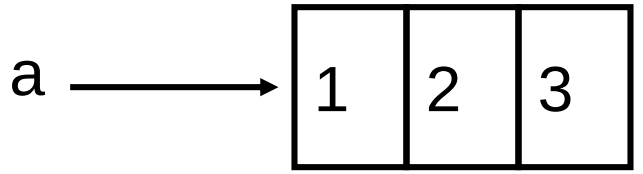
Example:

```
a=[1,2,3]
b=a
print("b=",b)
a.append(4)
print("b=",b) # refernce to object
```

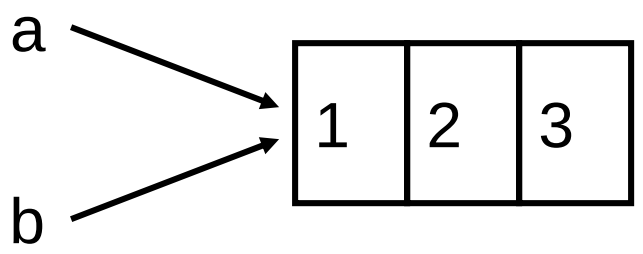
```
b= [1, 2, 3]
b= [1, 2, 3, 4]
```

CHANGING A SHARED LIST

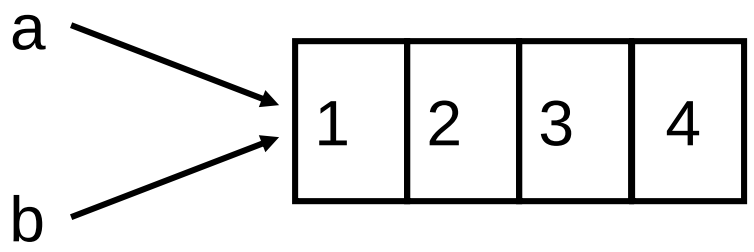
a = [1, 2, 3]



b = a

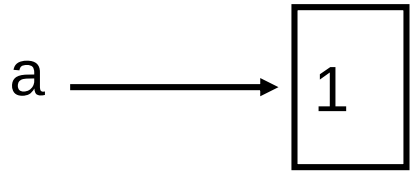


a.append(4)

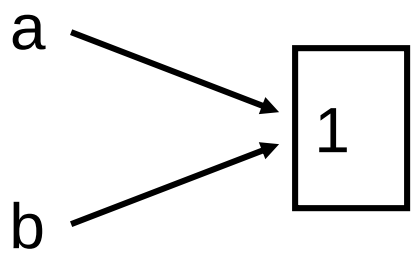


CHANGING AN INTEGER

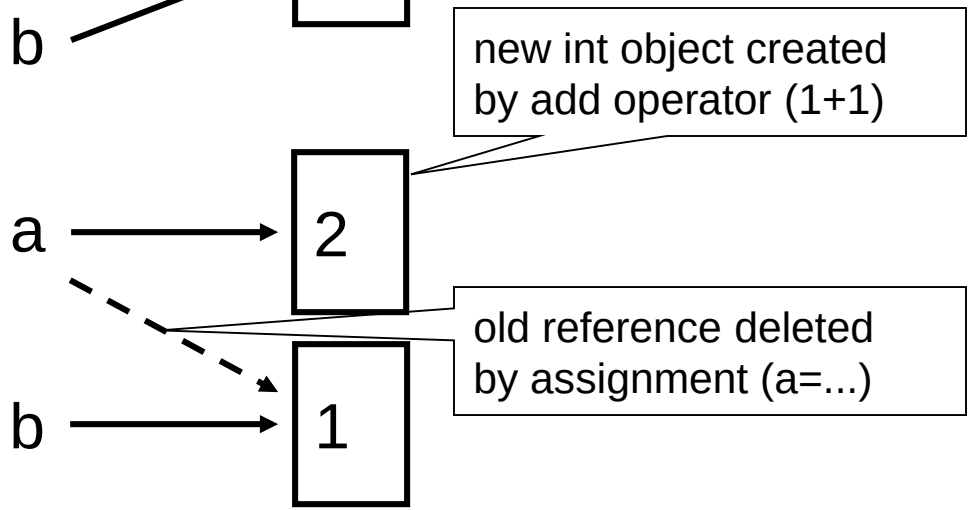
a = 1



b = a



a = a+1



CONTROL STRUCTURES

if *condition*:

    statements

[elif *condition*:

    statements] ...

else:

    statements

while *condition*:

    statements

for *var* in *sequence*:

    statements

break

continue

GROUPING INDENTATION

In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print("Bingo!“)
    print("---“)
```

In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n");
    }
      }
      printf("---\n");
}
```

FUNCTIONS

Syntax

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Example

```
def printme( str ):
   "This prints a passed string into this function"
   print str
   return
```

FUNCTIONS

Calling a Function: printme("hello")

Function Arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

return statement

https://www.tutorialspoint.com/python/python_functions.htm

EXERCISE

Using List or dictionary, write a program that calculates CGPA of the previous semester