



# Tutorial 9: Python - Basics

CS 104/108

Spring, 2024-25

TA: Mohana Evuri

Credits: Sabyasachi S. (2023 - 24 TA)

# Practice Problems apart from the Labs

- Extra practice problems (with no proper solutions/autograders) are put up on BodhiTree.

## Software Lab

[Multimedia Book](#) [Labs](#) [Announcements](#) [Interaction](#) [Performance](#) [About Course](#)

### Course Content

1. Course Logistics >

2. Unix Command Line >

3. Web Development >

4. Code Management >

5. Programming >

6. Practice Problems >

### Practice Problems

HTML, CSS and Javascript

GDB

Make

Bash

Sed and Awk

Basic Python




# Topics

- Variables
- Operations
- Strings
- Collections
- Conditionals and Loops
- Functions
- Class
- Modules
- File Handling

# Print Function

- The `print()` function is used to display information on the terminal/console.
- Arguments can be strings, numbers, or any objects. They are converted to the "str" datatype before printing.
- By default, multiple arguments are printed space-separated and terminated with a newline ("\n").
- The `sep` parameter can be used to change the separator between printed objects.
- The `end` parameter can be used to change the line ending.
- `print.py` is provided.

 print.py

```
1  # Printing multiple objects
2  name = "Alice"
3  age = 30
4  height = 5.8
5  print("Name:", name, "Age:", age, "Height:", height)
6
7  # Demo using sep and end with multiple objects
8  print("One", "Two", "Three", sep=", ", end="!\n")
```

```
Name: Alice Age: 30 Height: 5.8
One, Two, Three!
```

# Intro - Variables, Comments



- Variables are labels assigned to refer to a value or class object
- Python being an intelligent language, is dynamically-typed, i.e, variables need not be explicitly declared and can be overridden with other types, unlike C/C++.
- Nevertheless, casting can be used to specify the data-type
- Single-line comments start with `#`.
- Multi-line comments can be enclosed within triple quotes (`'''` or `"""`).
- **Rule:** Variable names can only contain letters, numbers and underscore. No spaces!
- **Advice:** Avoid using keywords as variable names: Eg: final, int, float
- **Note:** There is no concept of const in Python

# Handling Variables

- vars.py is provided.

```
## Dynamicness of Python's variables
```

```
x = 5
print("Initial x:", x, "Type:", type(x))

x = "Hello"
print("Updated x:", x, "Type:", type(x))
```

```
Initial x: 5 Type: <class 'int'>
Updated x: Hello Type: <class 'str'>
```

```
## Multiple Assignment
```

```
name, age, city = "John", 25, "New York"
print("Name:", name, "Age:", age, "City:", city)
```

```
## Casting in Python
```

```
a = str(3)      # a will be '3'
# b = int("3.5") # error
b = int(float("3.5")) # b will be 3
c = float("2")   # c will be 2.0

print("a:", a, "Type:", type(a))
print("b:", b, "Type:", type(b))
print("c:", c, "Type:", type(c))
```

```
a: 3 Type: <class 'str'>
b: 3 Type: <class 'int'>
c: 2.0 Type: <class 'float'>
```

```
Name: John Age: 25 City: New York
```

# Arithmetic Operators



- There are several mathematical operations supported by Python.
- Compound Operations refer to expressions that involve a combination of arithmetic operations and assignment in a more concise way.
- A compound operation  $a += 2$  is the same as  $a = a + 2$
- $5//2 = 2$
- Refer to [operators.py](#) for some simple and compound operations.

%	Modulus	^	Bitwise XOR
**	Exponentiation	~	Bitwise NOT
//	Integer division	<<	Left Shift
&	Bitwise AND	>>	Right Shift
	Bitwise OR		

# Strings



- Strings in Python can be declared using single (') or double (") quotes. You can exploit this flexibility.
  - "I'm Mohana and I am poor"
  - 'I am the poorest "TA" in CS104 ;('
- Escape Sequences: Escape sequences are special characters preceded by a backslash (\) in a string.
  - \n: New Line
  - \': Single Quote
  - \": Double Quote
  - \\: Back Slash
  - \t: Tab Space
  - \b: Backspace (Note: Its behavior might differ between Python 2 and Python 3)



# String Methods



- `+` operator to concatenate strings.
- `format()` method for string formatting.
- `upper()` and `lower()` to convert into upper and lower case.
- `strip()` to remove leading and trailing whitespace.
- `split()` to split a string into a list of substrings.
- `replace()` to replace a substring with another.
- `join()` to join elements of a list into a single string.
- Refer to [strings.py](#)

```
>>> message = " Python is great! "
>>> message = message.strip()
>>> message
'Python is great!'
>>> words = message.split()
>>> words
['Python', 'is', 'great!']
>>> words[-1] = "awesome"
>>> message = " ".join(words)
>>> message
'Python is awesome'
>>> message = message.replace("awesome", "amazing")
>>> message.lower()
'python is amazing'
>>> message.upper()
'PYTHON IS AMAZING'
>>> message = "Message is {}".format(message)
>>> message
'Message is Python is amazing'
```

# Lists



- Lists are ordered, mutable collections in Python.
- Elements in a list can be of different data types.
- Lists can be created using square brackets `[]`.
- Lists are versatile data structures in Python, allowing storage and manipulation of collections of items.
- Lists support various operations such as indexing, slicing, updating, and more.
- Negative indices can be used to access elements from the end of the list.
- `len()` function provides the length of the list.
- List methods include `append()`, `insert()`, `remove()`, `pop()`, `sort()`, `reverse()`, and `clear()`.
- List comprehension is a concise way to create lists based on conditions.
- Refer the `lists.py` file, in the shared folder.

# Tuples and Sets



- Tuples are immutable and suitable for fixed collections.
- Use tuples when you want to represent data that should not be changed.
- Sets are mutable and allow dynamic modification.
- Sets are useful for managing unique elements and performing set operations.
- **Tuple Methods:**
  - **count(x)**: Returns the number of occurrences of element **x** in the tuple.
  - **index(x)**: Returns the index of the first occurrence of element **x** in the tuple.
- **Set Methods:**
  - **add(x)**: Adds element **x** to the set if it is not already present.
  - **remove(x)**: Removes element **x** from the set. Raises an error if **x** is not present.
- Refer the **tuples\_sets.py** file in the shared folder

# Dictionaries



- Dictionaries are unordered collections of key-value pairs.
- Keys are unique and immutable; values can be of any data type.
- Created using curly braces `{}` or the `dict()` constructor.
  - Example: `grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'C'}`.
- `dict[key]` or `dict.get(key)` returns the value associated with the key.
- Elements are added using the syntax `dict[key] = value`.
- Existing elements can be modified by assigning a new value to the key.
- Elements are removed using the `pop(key)` method.
- `for key in dict` or `for key in dict.keys()` explicitly iterates over keys
- `for value in dict.values()` iterates over values
- `for key, value in dict.items()` iterates over key-value pairs
- The `len(dict)` function returns the number of key-value pairs.
- Refer the `dict.py` file in the shared folder

# Booleans and Conditional Operators



Booleans are either True or False. In Python, expressions are evaluated and one of the booleans are returned.

TRY: What does type casting bool on another data type do? For eg: `bool(27)` or `bool("TUB")`

Conditional Operators:

- `==` (Is Equal To?)
- `!=` (Is Not Equal To?)
- `>` (Is Greater Than?)
- `<` (Is Less Than?)
- `>=` (Is Greater Than and Equal To?)
- `<=` (Is Less Than and Equal To?)

```
>>> 2 > 3
False
>>> 2 != 10
True
>>> (18 + 2) == (22 - 2)
True
>>> 90 <= 90
True
```

# Logical and Membership Operators



Logical Operators are used to combine boolean expressions

- `and` : Returns True iff both expressions are True
- `or` : Returns True if either of the expressions are True
- `not` : Returns the negation of the evaluated expression

```
>>> (4 > 3) and (78 < 2)
False
>>> ((4 == 3) or (2 > 1)) and not (4>2)
False
```

Membership operators are used to test if the LHS is present in the RHS.

- `x in y` : Returns true if x is present in y
- `x not in y` : opposite of above

```
>>> x = [1,2,3,4,5]
>>> 2 in x
True
>>> 6 not in x
True
>>> a = [[1,2],3]
>>> 1 in a
False
>>> [1,2] in a
True
```

# Conditionals



- The **if** statement is used to execute a block of code only if a specified condition is true.
- The **else** statement is used to execute a block of code if the preceding **if** condition is false.
- The **elif** (else if) statement is used to check multiple conditions sequentially.
- Conditional statements can be nested, allowing for more complex decision-making.
- Take care of indentation.
- Refer to **conditionals.py**, toggle values of x,y

```
1  x = 10
2  y = 5
3
4  if x > 5:
5      if y > 3:
6          print("Printing A")
7      else:
8          print("Printing B")
9  elif x > 3:
10     if y > 4:
11         print("Printing C")
12     elif y > 2:
13         print("Printing D")
14     else:
15         print("Printing E")
16 else:
17     print("Printing F")
```

```
aria@aria-IdeaPad-Slim-5-14IAH8:~/Desktop/CS108/Python-I$ python3 conditionals.py
Printing A
```

# Range function

- The `range()` function in Python is used to generate a sequence of numbers. It can be used in a `for` loop to iterate over a sequence of numbers.

## Using `range(start, stop):`

```
python

for j in range(2, 8):
    print(j, end=' ')
```

• Output: `2 3 4 5 6 7`

## Using `range(stop):`

```
python

for i in range(5):
    print(i, end=' ')
```

• Output: `0 1 2 3 4`

## Creating a List using `range`:

```
python

numbers = list(range(5))
print(numbers)
```

• Output: `[0, 1, 2, 3, 4]`

## Using `range(start, stop, step):`

```
python

for k in range(1, 10, 2):
    print(k, end=' ')
```

• Output: `1 3 5 7 9`



# Special Keywords



## continue:

- Skips the rest of the code inside a loop for the current iteration and move to the next iteration.
- It is often used to skip certain conditions and continue with the next iteration.


## break:

- Exits the loop prematurely, regardless of the loop condition.
- It is commonly used to terminate a loop when a specific condition is met.

python

```
for i in range(5):  
    if i == 2:  
        continue  
    print(i)
```

Output:


 Copy code

0  
1  
3  
4

python

```
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

Output:

 Copy code

0  
1  
2

# Functions

- A function is a reusable block of code that performs a specific task. They help organize code and promote reusability.
- Functions are declared using the `def` keyword, followed by the function name and parameters.
- Parameters are variables listed in the function definition.
- Arguments are the actual values passed to the function when it is called.
- Functions can return values using the `return` statement.
- If no return statement is present, the function returns `None` by default.
- Functions are called by using the function name followed by parentheses `()` containing the arguments

```
def solve_linear(a,b):  
    """  
        Solves the linear equation  $a*x + b = 0$ .  
        Takes two arguments, a and b, and returns the solution x.  
    """  
    if a != 0:  
        return -b/a  
    else:  
        print("No solution")  
        return None
```

```
aria@aria-IdeaPad-Slim-5-14IAH8:~/Desktop/CS108/Python-I$ python3  
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> from solver import solve_linear  
>>> solve_linear(2,3) #solution to  $2x+3 = 0$   
-1.5  
>>> solve_linear(0,1) #solution to  $0x+1 = 0$   
No solution  
>>>
```

# Functions

```
def func(a):  
    var1 = a*a # local scope var1  
    var2 = 10  
    f = var1 + 2  
  
    global var3 # global accesses the globally defined var3  
    var3 = 10 # var3 modified in global scope  
  
    return var1  
  
var1 = 20  
var3 = 8  
print("Before running func", var1, var3)  
print("func output", func(2))  
print("After running func", var1, var3)  
# print(var2) # -- error
```

```
aria@aria-IdeaPad-Slim-5-14IAH8:~/Desktop/CS108/Python-I$ python3 scope.py  
Before running func 20 8  
func output 4  
After running func 20 10
```

- **Scope:**
  - Variables defined inside a function have local scope and are not accessible outside the function.
  - Variables defined outside a function have global scope and can be accessed throughout the program.
- Functions can have default parameter values, which are used if no argument is provided for that parameter.
- Refer to [solver.py](#) for some demo functions and [scope.py](#) to understand local and global scope.

# Class



- Python supports object-oriented programming (OOP) with the ability to define classes.
- Classes encapsulate data and methods that operate on that data.
- The `Student` class represents a student with attributes such as name, major, CPI, and credits.
- The `__init__` method initializes a student object with default values.
- The `__str__` method provides a string representation of the student.
- Class methods include `get_cpi`, `add_course`, and `branch_change`.

```
6  class Student:
7      def __init__(self, name, major):
8          self.name = name
9          self.major = major
10         self.cpi = 0
11         self.credits = 0
12
13     def __str__(self):
14         return self.name + ", " + self.major + " dept."
15
16     def get_cpi(self):
17         return self.cpi
18
19     def add_course(self, num_credits, grade):
20         self.credits += num_credits
21         self.cpi = (self.cpi*(self.credits - num_credits
22
23     def branch_change(self, branch_prefs):
24         new_major = bc(self.cpi, branch_prefs)
25         if new_major == None:
26             print("Sorry, you cannot branch change.")
27         else:
28             print("Congratulations! You successfully bra
29             self.major = new_major
30
31
```

# Objects

- Four student objects (`student1`, `student2`, `student3`, `student4`) are created with different majors.
- Whenever any student is printed e.g: `print(student1)`, showcases the string representation of each student.
- The object attributes can be accessed in this way: `student1.name`, `student2.cpi`.
- The class methods can be called using: `student1.add_course(arg1, arg2)`.
- Refer to `student.py`

```
student1 = Student("Rohan", "CSE")
student2 = Student("Sabya", "EE")
student3 = Student("Rohit", "ME")
student4 = Student("Sahil", "CE")
```

```
students = [student1, student2, student3, student4]

for student in students:
    print(student)
print("-----")
```

```
Rohan, CSE dept.
Sabya, EE dept.
Rohit, ME dept.
Sahil, CE dept.
-----
```

```
student2.add_course(6, 9)
student2.add_course(4, 8)
student2.add_course(8, 7)

print(f"{student2.name} has cpi {student2.cpi}")
print(f"{student2.name} is in {student2.major} dept.")
print("-----")
# Attempt branch change
branch_preferences = ["CSE", "EE", "ME", "CE", "CHE", "ENV", "BSBE"]
print(f"{student2.name} is attempting branch change.")
student2.branch_change(branch_preferences)
print(f"{student2.name} is in {student2.major} dept.")
print("-----")
```

# Modules



- A module is a file containing Python definitions and statements.
- Modules help organize code, making it more manageable and reusable.
- Python provides various built-in modules that offer functionalities for mathematics (`math`), random numbers (`random`), and timing (`time`).
- Python allows the creation of user-defined modules. Example: `solver.py`, `branch_change.py`.
- Refer to `solver.py`, `branch_change.py` and `modules.py`

# Files



- `open()`
  - a. Takes two parameters: filename and mode.
  - b. Filename is the name of the file to be opened.
  - c. Mode specifies the purpose of opening the file (read, append, write, create).
- **Modes:**
  - a. "r": Read (default). Opens a file for reading. Raises an error if the file does not exist.
  - b. "a": Append. Opens a file for appending. Creates the file if it does not exist.
  - c. "w": Write. Opens a file for writing. Creates the file if it does not exist.
  - d. "x": Create. Creates the specified file. Returns an error if the file exists.
- `read()` and `readline()`
  - a. Methods of the file descriptor to read the entire contents of the file and just one line of the file respectively
- Refer to `files.py`



# Exercises





## Problem Statement 1

Write a python script without using any external modules to generate output as shown in Fig.

```
aria@aria-IdeaPad-Slim-5-14IAH8:~/Desktop/CS108/Python-I$ python3 prob1.py
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
```



## Problem Statement 2

Write a function `matmul(A,B)` which takes two  $m \times n$  matrices A and B and returns the matrix product AB. Before performing the multiplication, do a quick sanity check to see if the product is feasible, if not return -1. Matrices are written in the list of list format, with inner lists corresponding to the same row.

```
aria@aria-IdeaPad-Slim-5-14IAH8:~/Desktop/CS108/Python-I$ python3 prob2.py
A:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
B:
[3, 1, 2]
[9, 10, 1]
[4, 5, 6]
C = AB:
[33, 36, 22]
[81, 84, 49]
[129, 132, 76]
```

Help: Implement the  
TODO in the  
[prob2\\_helper.py](#)



## Problem Statement 3

You are given a folder named students, which has information about the courses undertaken and grade scored, for each student in different file. Your task is to create a function, which takes in two arguments, the file path and branch change preferences, and prints the results of branch change on terminal.

Hint: You can import branch\_change function

Help: [prob3\\_helper.py](#) provided

```
aria@aria-IdeaPad-Slim-5-14IAH8:~/Desktop/CS108/Python-I$ python3 prob3.py
Branch change attempted for student Sahil
Branch Preferences: ['CSE', 'EE', 'ME']
CPI: 8.285714285714286
Sorry, you cannot branch change.
```



## Additional Practice

**Implement the K Means Algorithm.**

```
def kmeans(data, K):  
    labels = None  
    centroids = None  
    ### TODO  
    return labels, centroids
```

**Note:** The algorithm is not part of the syllabus. This assignment is just to test your ability to convert pseudo-codes into actual running code.

Algorithm: <https://www.youtube.com/watch?v=4b5d3muPQmA>

Solution: <https://www.geeksforgeeks.org/k-means-clustering-introduction/>



**Thank You 🧩 !!!**