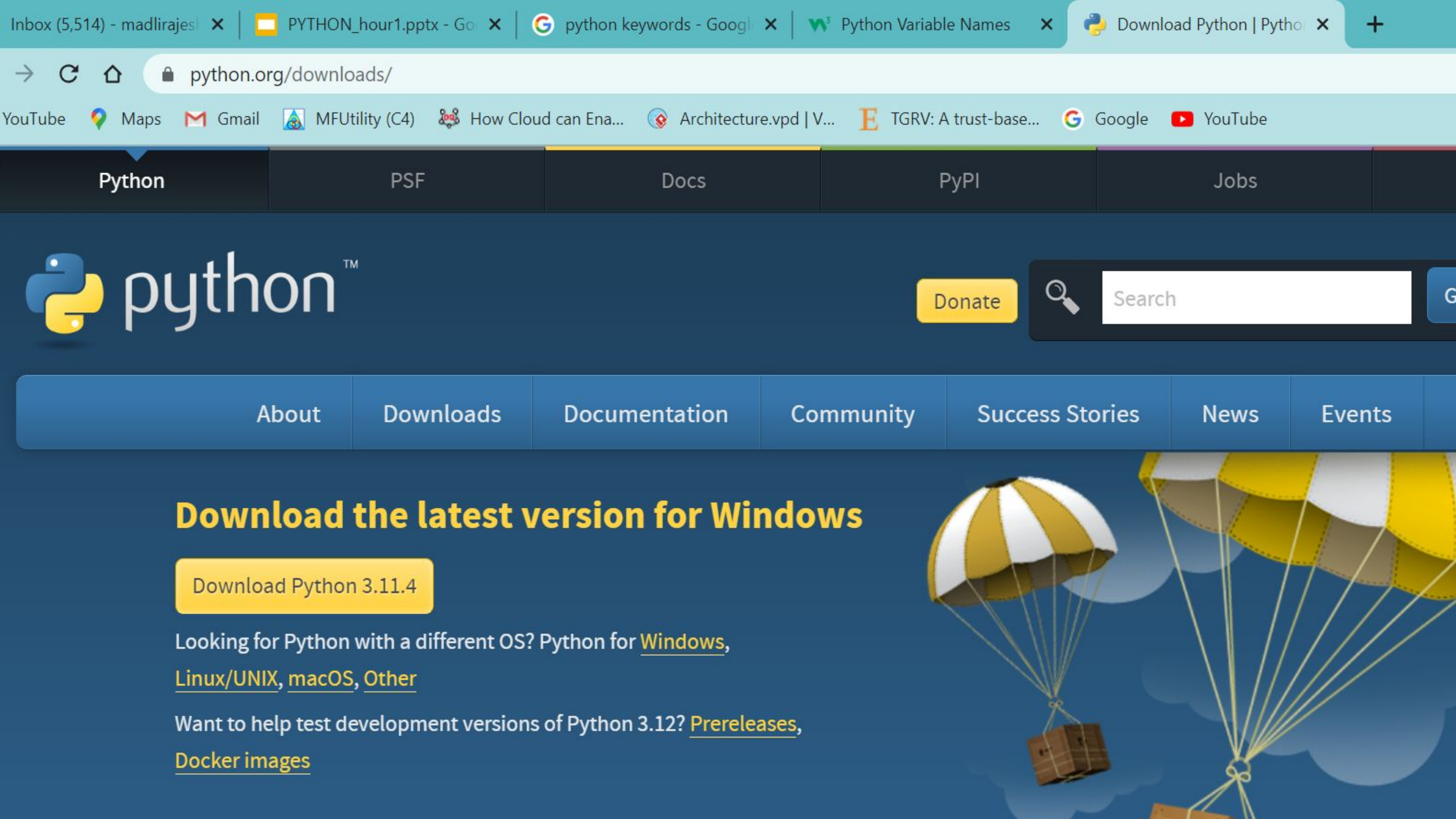
Unit-1: Python Basics

Prof. Rajeshwari Madli
Assistant Professor
Dept. of CSE

Python Basics

- Python is a interpreted language, whose main design philosophy was to develop a more readable program.
- It allows users to write programs in fewer lines compared to other languages.
- It is dynamically typed and garbage collected.
- The stable version of Python is 3.11



Download the latest version for Windows

Download Python 3.11.4

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.12? Prereleases, Docker images

What can go wrong?

- Syntax errors
- Logical errors/Semantic errors
- Runtime errors

Variables, expressions and statements

- **Values**
- A value is any data that can be used in a python script.
- It can be an integer, decimal value or a character value.
- Ex: 10, 5.676, ‘a’, ‘Hello’ etc.

- **Types**
- The type the value belongs to, can be found using the function type().
- Ex: type(25)
- type(‘a’)
- type(4.564)

- **print statement**
- It is a output statement in Python.
- Ex: print(25)
- print(25,00,000)

Variables

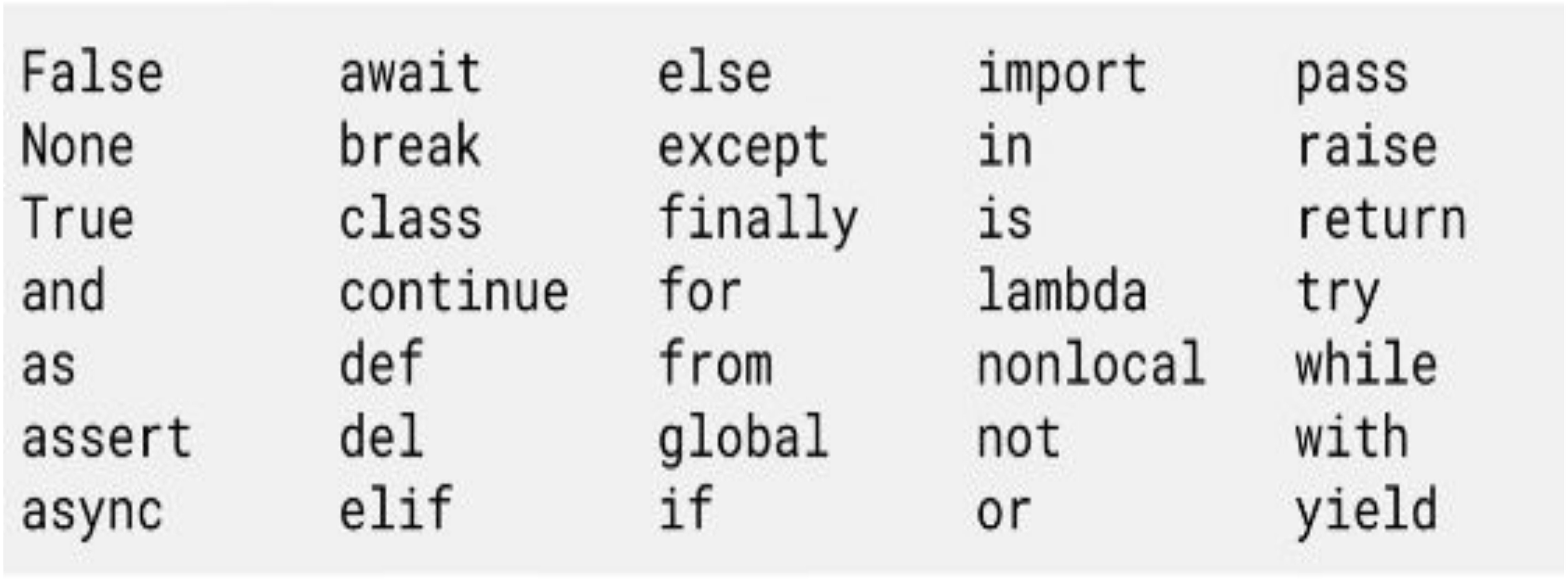
- A variable is name that refers to a value, that can be manipulated at any point in the program.
- Ex:
- 1) Msg = “Good Morning”
- print(Msg)
- 2) x=10
- x=x+10

Variable names and Keywords

- **Rules**
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Keywords cannot be used as variables.

Keywords

- Keywords are reserved words.
- Python reserves 35 keywords.



False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Identify Valid and Invalid variable names

- 1. _integer
- 2. 1_marks
- 3. marks-1
- 4. class1
- 5. TrueFalse
- 6. False
- 7. @unit

8. as_in
9. length*breadth
10. _2_
11. Return
12. all@1
13. First.Name
14. Last_Name
15. a2b

Statements

- It is a unit of code that the python interpreter can execute.
- A python script is a sequence of statements.
- Ex:
- Msg=“Hello, Good Morning”
- print(Msg)

Operators and Operands

- An **operator** is a special symbol that performs some computation, such as addition, subtraction and multiplication.
- Ex: +,-,*,/,**,//
- An Operand is value or a variable on which the operator operates.

Expressions

- An expression is a combination of values, variables and operands, that produces some output.
- - Ex:
- 10+20
- 5/10
- x=10
- x=x**3
- Floored division //

Order of operations

- When an expression contains more than one operator, the order of evaluation depends on the rules of precedence.
- Precedence is the priority of operators.
- For mathematical operations, Python follows PEDMAS rule(Parenthesis, Exponentiation, Division, Multiplication, Addition, Subtraction)

1. (4-2)*(5-2)
2. (1+1)**(5-2)
3. (4*5/2)+(4-3+1)

Modulus Operator

- The modulus operator is used to obtain the remainder of division operation.

Ex: 20%3

- Rem= 20%3
- Quo=20//3
- Rem1=2345%10
- Rem2=2345%100

String Operations

- The addition operator + works with strings.
- When used with strings, it performs concatenation.
- Ex: first=100
- Second=200
- print(first+second)
- first=‘100’
- Second=‘200’
- print(first+second)

- The multiplication operator * multiplies the content of a string by an integer.
- Ex: x=“Hello”
- print(x*3)

File Creation and Execution in IDLE

Asking input from the user

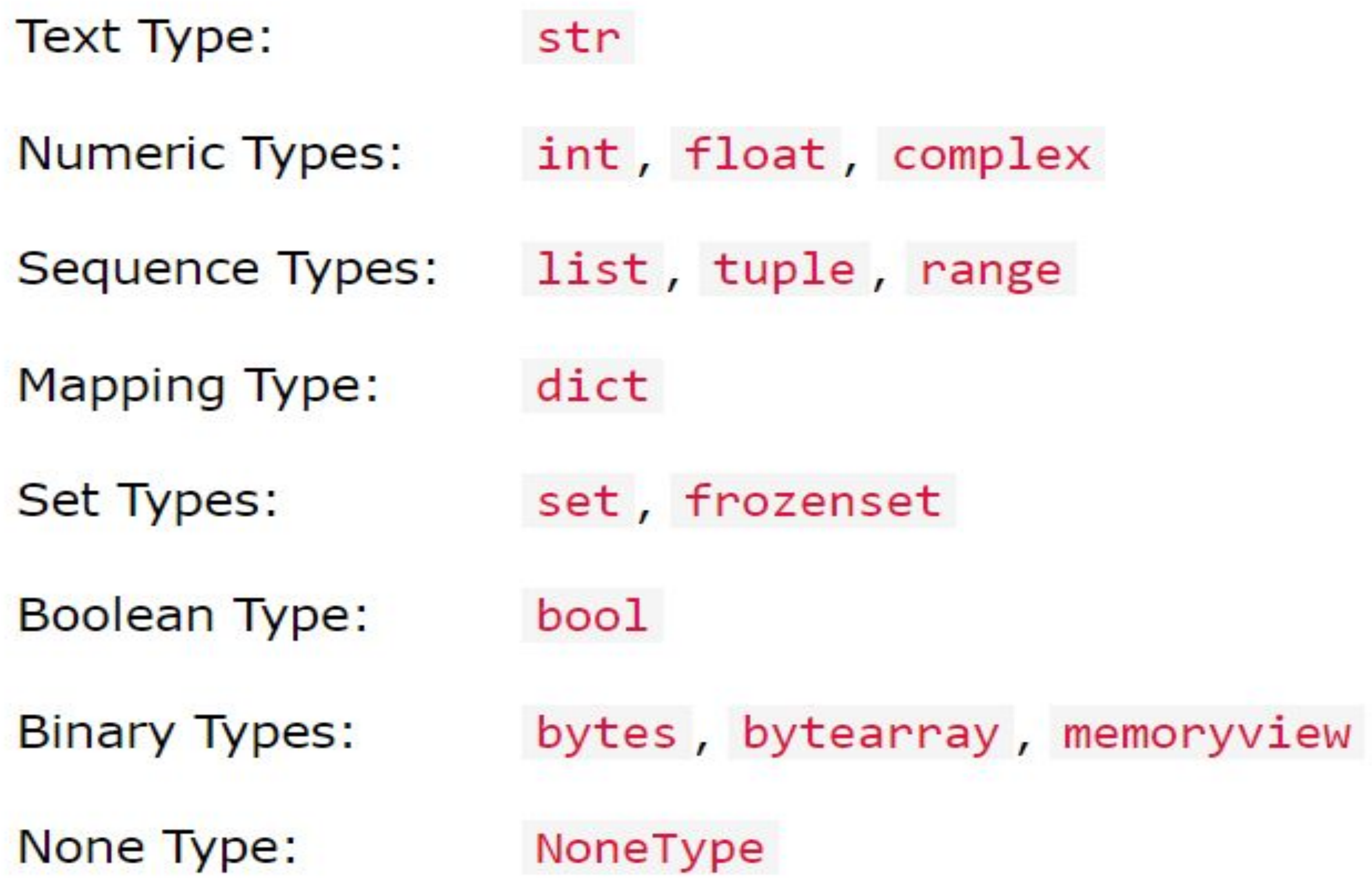
- **input()** function in python helps to read the input from the user.
- When this function is called, the program waits for the user to enter an input through keyboard. When the ENTER key is pressed, the program resumes.
- Ex: inp= input(“What is your name?”)

By default, whatever the user enters from the keyboard, is taken as a string.

Choosing mnemonic variable names

- Mnemonic- means “memory aid”
- Example a variable names:
  - Abdfs
  - X1
  - Number1
  - Length

Data Types in Python



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`
None Type:	`NoneType`

Examples

- x = 35e3
  y = 12E4
  z = -87.7e100
- x=5j
- x=4-5j
- x = ["apple", "banana", “mango"]
- x = ("apple", "banana", "cherry")
- x = {"name" : "John", "age" : 36}
- x = {"apple", "banana", "cherry"}
- x = True

Type Conversion

- Converting the data from one type to another.
- Syntax

Data-Type(Value/Variable/Object)

Sample Programs

- Write a python program to read 2 input numbers from users and perform all arithmetic operations on them.
- Write a python program to read the user’s first name, middle name and last name, and print them using concatenation operator.
- To find the square root of a given number.
- To swap the contents of 2 variables.

Conditional Execution

- Ability to check conditions and change the program behavior.
- A particular statement is executed only if some condition is true. Also called as decision making statements.
- Conditions are Boolean Expressions.
- **Boolean Expressions:** An expression that evaluates to either True or False.
- A combination of **relational and logical operators** are used in Boolean expressions.

Relational Operators

- Check the relation between 2 operands.
- **x == y   # Is x equal to y?**
- **x != y   # x is not equal to y**
- **x > y   # x is greater than y**
- **x < y   # x is less than y**
- **x >= y   # x is greater than or equal to y**
- **x <= y   # x is less than or equal to y**
- **x is y   # x is the same as y**
- **x is not y   # x is not the same as y**

Logical Operators

- Generally used to combine 2 or more relational expressions.
- 3 logical operators are supported by python
    - **and**
    - **or**
    - **not**

Ex:

1. x>y **and** x>z
2. True **and** False
3. **not** (x>y)

4 types of Conditional statements

1. Simple **if**
2. Alternate execution- **if else**
3. Chained conditionals- **if elif**
4. Nested conditionals

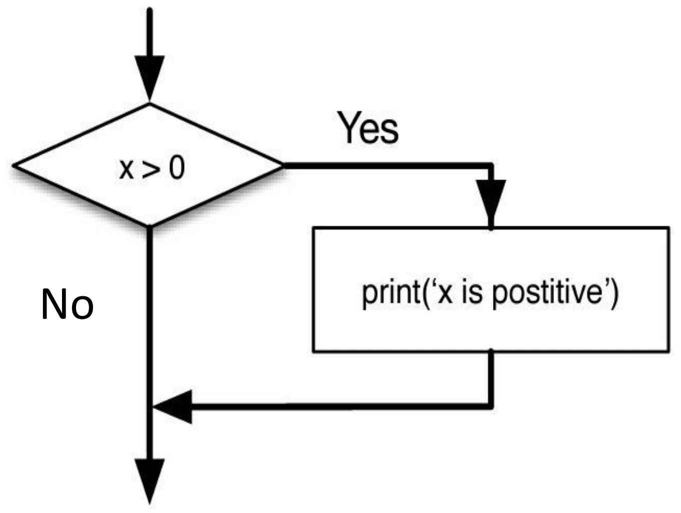
1. Simple if

- **Syntax:**

```
if condition1:
    Statements   #body of if
```

- Example:

```
x=10
if x>0 :
    print("x is positive")
```



- The boolean expression after the if statement is called the ***condition.*** We end the **if** statement with a colon character **(:)** and the lines after the **if** statement are indented.

pass statement

```
if x>y :
    pass
```

- Predict the output

```
x=11
if x>10:
    print('Expression was true')
    print('Done.')
print('After conditional')
```

2. Alternate Execution- if elif

- **<u>Syntax</u>**

```
if condition:
    statements block1
else:
    statement block2
```

Example: Program to check if the given number is a multiple of 5 and 8, or not.

Programs on conditional statements

1. Write a python program that will check for the following conditions:
    - If the light is green – Car is allowed to go
    - If the light is yellow – Car has to wait
    - If the light is red – Car has to stop
    - Other signal – unrecognized signal. Example black, blue, etc…
2. Write a program to check students' grades. Your program should fulfill the following conditions:
    - Grade A – Outstanding
    - Grade B – Excellent
    - Grade C – Very Good
    - Grade D – Good
    - Grade E – Satisfactory
    - Grade F– UnSatisfactory

The program should also ask to enter the student's name, class, and section.

3. Modify the earlier program (3rd Program) students' grades in such a way that they should take in five subject marks. Find the total mark and their percentage. Your program should check for the following conditions:

- If the percentage falls below 40, they are considered fail.
- If the percentage is between 40 and 60, grade them as pass.
- If the percentage is between 60 and 75, grade them as good.
- If the percentage is between 75 and 85, grade them as very good.
- If the percentage is between 85 and 100, grade them excellent.
- If the percentage is below zero or above 100, it's an error.

Note: Demonstrate nested condition

4. Write a python program to find the final bill to be paid by the customer. Customers are offered discounts based on the mode of payment. (Debit card- 10% disc, Credit card – 20%, UPI – 30%).

Catching Exceptions using try and except

- Consider a sample program to convert a Fahrenheit temperature to a Celsius temperature

```
inp = input('Enter Fahrenheit Temperature: ')

fahr = float(inp)

cel = (fahr - 32.0) * 5.0 / 9.0

print(cel)
```

- There is a conditional execution structure built into Python to handle these types of expected and unexpected errors called “**try / except**”.
- The idea of try and except is that you know that some sequence of instructions may have a problem and you want to add some statements to be executed if an error occurs. These extra statements (the except block) are ignored if there is no error.

```
inp = input('Enter Fahrenheit Temperature: ')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:     #catching an  exception
    print("enter a number")
```

Short circuit evaluation of Logical Expressions

- When Python is processing a logical expression such as

x=5,y=4

 x >= 2 and (x/y) > 2

It evaluates the expression from left to right.

- Because of the definition of **and**, if x is less than 2, the expression x >= 2 is False and so the whole expression is False regardless of whether (x/y) > 2 evaluates to True or False.

1.

x = 6

y = 2

x >= 2 and (x/y) > 2

2.

x = 1

y = 0

x >= 2 and (x/y) > 2

3.

x = 6

y = 0

x >= 2 and (x/y) > 2

Iterations

- Iterative statements allow a similar set of statements to be executed again and again until a certain condition is satisfied.
- Also called as Loops.
- Updating Variables

1. *x=x+2*

print(x)

2. x=2

x=x+2

print(x)

1. while loop

- The Syntax of While Loop is:

```
while <expr>:
    <statement(s)>
```

Ex:

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Hurray!')
```

Flow of execution for a while statement:

1. Evaluate the condition, yielding True or False.
2. If the condition is false, exit the while statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1.

Infinite loops

- Suppose you write a while loop that theoretically never ends

```
1.  while True:
     print('foo')
2. x=10
    while x>1:
     print(‘Hi’)
     x=x+1
```

3.

```
n = 10
while True:
    print(n, end=' ')
    n = n - 1
print('Done!')
```

4.

```
n = 10
while True:
    inp=input(“Enter a line:”)
    if inp == “End”:
        break
    print (line)
print('Done!')
```

Examples

- Write a python program to find the factorial of a given number.
- Write a python program to find the sum of first n natural numbers.
- Find the sum of digits of a given number.
- Check if the given number is palindrome or not.
- Write a program to find the greatest common divisor of the given 2 numbers.

break and continue statements

- The break statement stops the execution of the loop and the control goes out of the loop.
- The continue statement is used to skip the current iteration and the control goes to the next iteration.

```
N=1
while N<=5
    if N==3:
        N=N+1
        continue
    print(N)
    N=N+1
```

Definite loops using for

For loop is called as definite loop because if allows us to iterate through a finite set of objects.

General syntax:

```
for <var> in <iterable>:
        <statement(s)>
```

1.

```
a = [‘apple', ‘banana', 'berry']
for i in a:
   print(i)
```

2.

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
     print('Happy New Year:', friend)
print('Done!')
```

Counting and Summing Loops

- If we want to count the number of items in the collection of objects, we can write it as follows:

```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Count: ', count)
```

- Summing

```
sum = 0
for var in [3, 41, 12, 9, 74, 15]:
    sum = sum + var
print(‘Sum: ', sum)
```

Maximum and Minimum loops

- Finding the maximum number in the collection of objects.

```
largest = None
print('Before:', largest)
for var in [3, 41, 12, 9, 74, 15]:
    if largest is None or var > largest :
        largest = var
print('Largest:', largest)
```

range() function in python

- The range() function in python returns a sequence of values in the given range.
- It can take 1/2/3 arguments (min:1 and max:3)
- *range(stop)*
- *range(start:stop)*
- *range(start:stop:step)*

Examples

1.

```
for i in range(6):
    print(i, end=" ")
```

2.

```
for i in range(5,20):
    print(i, end=" “)
```

3.

```
for i in range(5,50,5):
    print(i, end=" ")
```

```
fact=1
for i in range(1, 5)
    fact=fact*i
print(fact)
```

Write Programs

- Write a Python program to print all the even numbers within the given range.
- Write a Python program to calculate the sum of all numbers from 1 to a given number
- Write a Python program to calculate the sum of all the odd numbers within the given range
- Write a Python program to count the total number of digits in a number.
- Write a program to print multiplication tables within a given range.
- Write program to print the given pattern

```
1 2 3 4 5

1 2 3 4

1 2 3

1 2

1
```