INTRODUCTION TO PYTHON

ORIGIN

- **PYTHON** IS A GENERAL-PURPOSE INTERPRETED, INTERACTIVE, OBJECT-ORIENTED, AND HIGH-LEVEL PROGRAMMING LANGUAGE. IT WAS CREATED BY GUIDO VAN ROSSUM DURING 1985-1990. LIKE PERL, PYTHON SOURCE CODE IS ALSO AVAILABLE UNDER THE GNU GENERAL PUBLIC LICENSE (GPL).

Python

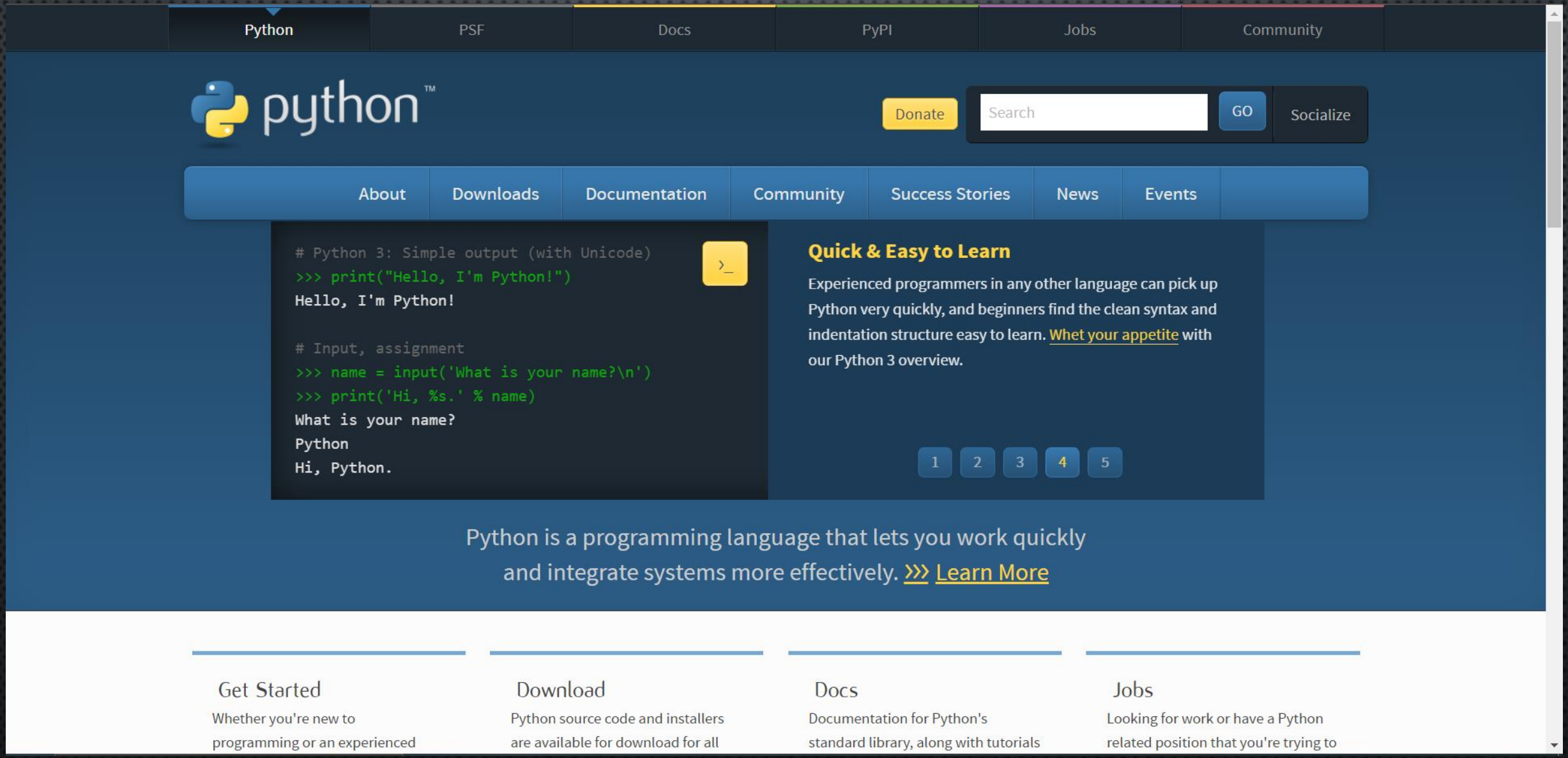
Python is a widely-used programming language because it's an easy to learn language while still maintaining a wide variety of applications including:

- Data science and machine learning
- Automation
- Web
- security

Benefits of using Python

- Interpreted language: the language is processed by the interpreter at runtime, like PHP or PERL, so you don't have to compile the program before execution.
- Interactive: you can directly interact with the interpreter at the Python prompt for writing your program.
- Perfect for beginners: for beginner-level programmers, Python is a great choice as it supports the development of applications ranging from games to browsers to text processing.

HTTPS://WWW.PYTHON.ORG/



GO TO DOWNLOADS

Download the latest version for Windows

Download Python 3.9.6

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

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

Looking for Python 2.7? See below for specific releases

Active Python Releases

For more information visit the Python Developer's Guide.

Python version	Maintenance status	First released	End of support	Release schedule
3.9	bugfix	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537
3.6	security	2016-12-23	2021-12-23	PEP 494
2.7	end-of-life	2010-07-03	2020-01-01	PEP 373

Configure Paths

Installing Python on Ubuntu

- **Option 1: Install Python 3 Using apt (Easier)**
- This process uses the **apt package manager** to install Python. There are fewer steps, but it's dependent on a third party hosting software updates. You may not see new releases as quickly on a third-party repository.
- Most factory versions of Ubuntu 18.04 or Ubuntu 20.04 come with Python pre-installed.

OPTION 1 STEPS

- sudo apt update
- sudo apt install software-properties-common
- sudo add-apt-repository ppa:deadsnakes/ppa
- sudo apt update
- sudo apt install python3.8

Option 2 steps

- sudo apt update
- sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev wget
- cd /tmp
- wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
- tar -xf Python-3.7.5.tgz
- cd python-3.8.3
- ./configure --enable-optimizations (can take upto 30 minutes)
- sudo make altinstall
- sudo make install

WSL (OPTIONAL)

- If you would like to keep your windows or you don't want dual boot, instead you can use wsl.
- This link shows how you can download it and use it.

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, TESTVARIABLE and testvariable are two different identifiers in Python.

Conventions for identifiers

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

RESERVED WORDS

- PYTHON LIKE ANY OTHER PROGRAMMING LANGUAGE HAS RESERVED WORDS THAT CAN'T BE USED FOR A VARIABLE OR CONSTANT.

Lines and Indentation

- Unlike c or c++ , python doesn't have curly brackets to use with loops and conditions to indicate which part of the code belongs to which, instead we use indentation (spaces).
- A block of code in python is defined by the spacing (line indentation)
- in Python all the continuous lines indented with same number of spaces would form a block.
- Example:

```
if True:
    print "True"
else:
    print "False"
```

EXERCISE

```
IMPORT SYS
TRY:
    F = OPEN('TEST.TXT', "W")
EXCEPT IOERROR:
    PRINT "ERROR WRITING", F
    SYS.EXIT()
PRINT "ENTER '", FILE_FINISH,
PRINT "YES FILE FINISHED"
WHILE FILE_TEXT != FILE_FINISH:
    FILE_TEXT = RAW_INPUT("ENTER TEXT: ")
    IF FILE_TEXT == FILE_FINISH:
        F.CLOSE
        BREAK
    F.WRITE(FILE_TEXT)
    F.WRITE("\N")
F.CLOSE()
FILE_NAME = RAW_INPUT("ENTER FILENAME: ")
IF NOT LEN(FILE_NAME):
    SYS.EXIT()
TRY:
    FILE = OPEN('TEST.TXT', "R")
EXCEPT IOERROR:
    PRINT "THERE WAS AN ERROR READING FILE"
```

ERROR EXAMPLE

```
if True:
print "True"
else:
print "False"
```

MULTI-LINE STATEMENTS

- `TOTAL = CAR + \`
- `HOUSE + \`
- `FOOD`
- Code written within parenthesis or brackets do not need to use the line continuation character.

```
YEARS= ['2010', '2011', '2012',
        '2013', '2014']
```

Comments

- Comments are a great way to document what you have written in your code.
- They are an essential part of any code you write especially when you need to revisit it.
- Comments in python for one line are done using #.

```
#First comment
```

- Multiline comments are using three single quotations at the beginning and the end.

```
'''
Print('hello')
Print('This is a test')
'''
```

Multiple Statements on a Single Line

- The semicolon (;) allows multiple statements on the single line as long as both statements don't start a new code block. example
- `import os; y = 'yazan'; print(y)`

WHERE TO WRITE THE CODE

- IDLE
- Terminal (Interpreter) . (don't use it for production because nothing will be saved)
- Vscode
- Jupyter notebook
- Google colab
- Pycharm

VARIABLES

1

Variable's name are case sensitive.

2

Variable's name should always start with a letter and cannot contain spaces.

3

Don't use quotations when printing a variable.

4

You can overwrite any variable with another value even if that value has a different type.

Assigning Values to Variables

- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example –

DATA TYPES

Numeric:

- Integer. (1 , 4 ,90)
- Float. (0.1, 7.91, 76.876)
- Complex. (1J)

Text:

- Strings. ("Test", "Hello ")

Mapping Type:

- Dictionary. ({"Key" : "Value"})

DATA TYPES

Sequence type:

- List. ([1, 2, 3], [67,22, 111, 190])
- Tuple. ((1,2,3), (67,22,111,190))
- Range. (x = range(6))

Set. ({"1", "2", 1 , 2})

Boolean. (True, False)

Object Oriented and Python

- In programming there is a concept called object-oriented programming where we have models / templates that hold certain characteristics and features for different data types these templates are called "Classes".
- Objects inherit the class's features.
- Python is an object-oriented programming language.

MATHEMATICAL OPERATIONS

- 5 + 7
- 7 – 9
- 8 * 8
- 6 / 4 (NORMAL DIVISION)
- 6 // 4 (INTEGER RESULT , GETS RID OF THE REMAINDER)
- 6 ** 2 (POWER OF)

OPERATORS

- 10 == 9
- 10 != 9
- 1 > 4
- 2 < 3
- 10 >= 9
- 9 <= 10
- NOT X = 'HELLO'
- '5' IN "I ATE 5 APPLES"

ASSIGNING VALUES TO VARIABLES

- X = 7
- X += 1
- X -= 1
- X *= 2
- X /= 3

Getting an input from user

- Getting an input from a user can be easily achieved by using ***input().***
- When receiving an input from the user , the input always returns a string.
- Before performing any mathematical operation we need to perform type conversion to convert the string we have into the datatype we need.

String operations

- You can concatenate two strings by adding them.
- List(string) transforms each character into an element in the list.
- Split separates the string according to the character you specify.
- Strip removes spaces from start and end.
- Join, transforms a list of strings into a single string that is separated by the character used to initiate the join.

String Indexing Example

```
str = 'Good Morning All!'

print str           # Prints entire string
print str[0]        # Prints the first letter
print str[2:5]      # Prints letters from 3rd to 5th
print str[2:]       # Prints letters from 3rd character till the end
print str * 2       # Prints string two times
print str + "second string test" # Prints concatenated string
```

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

TRIPLE QUOTES

- PYTHON'S TRIPLE QUOTES ALLOWS STRINGS TO BE ON MULTIPLE LINES, INCLUDING VERBATIM NEWLINES, TABS, AND ANY OTHER SPECIAL CHARACTERS.

EXAMPLE:

```
S = """TEST LONG THAT IS MADE UP OF
TAB ( \T ) AND THEN ANY RANDOM WORD FOLLOWED BY THIS [ \N ], IT IS ALSO
GOOD FOR TEACHING STUDENTS WHAT IS THE PURPOSE BEHIND USING SUCH THINGS
LIKE TRIPLE QOUTES, THANK YOU VERY MUCH FOR TESTING.
"""
PRINT S
```

Lists

- Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

EXAMPLE

```
list = [ 'yazan', 111 , 222, 333, 'ahmad' ]
print list           # Prints complete list
print list[0]        # Prints first element of the list
print list[1:3]      # Prints elements starting from 2nd till 3rd
print list[2:]       # Prints elements starting from 3rd element
print list * 2  # Prints list two times
print list + list # Prints concatenated lists
```

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.

```
T= ( 'yazan', 111 , 222, 333, 'ahmad' )
print T          # Prints complete list
print T[0]       # Prints first element of the list
print T[1:3]     # Prints elements starting from 2nd till 3rd
print T[2:]      # Prints elements starting from 3rd element
print T * 2  # Prints list two times
print T + T # Prints concatenated lists
```

Tuples vs lists

- Indexes start at 0 for both lists and tuples.
- Tuples are immutable (the elements inside cannot be changed).
- Lists are mutable (can be changed).
- Lists have functions like append, pop. Reverse and sort.
- You can concatenate two lists or two tuples.

X = [1,2,3,4,5] (list)

X = (1,2,3,4,5) (tuple)

SETS

- Sets are like lists but it cannot have the same value twice , all values are unique values.
- You can use .add and .pop on sets.

X = {1,2,3,4,5} (set)

DICTIONARIES

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([])
- Contains keys and values, you call the key to get the value inside.
- You can change the values.
- Key names are unique , you can't have the same key name.
- You can add new keys.

Dictionaries

- You can use .clear to clear the dictionary
- You can use .keys to show all the available keys
- You can use pop with the key name to pop a certain value
- You can use items to list everything in the dictionary as a list of tuples.

EXAMPLE

```
DICT = {}
DICT['TEST'] = "TEST1"
DICT[1]      = "TEST2"

TINYDICT = {'NAME': 'ZAID','NUMBER':83647286, 'WORK': 'AI'}


PRINT DICT['ONE']       # PRINTS VALUE FOR THE SELECTED KEY
PRINT DICT[2]           # SAME AS THE ABOVE BUT DIFFERENT KEY
PRINT TINYDICT          # PRINTS COMPLETE DICTIONARY
PRINT TINYDICT.KEYS()   # PRINTS ALL THE KEYS
PRINT TINYDICT.VALUES() # PRINTS ALL THE VALUES
```

Data type conversion

- It is important to sometimes change data type like we said for example when getting an input from the user or for any other use case.
- To convert between types, you simply use the type name as a function.
- There are many options, you will find them in the next slide.

Sr.No.	Function & Description
1	int(x [,base]) Conv erts x to an integer. base specifies the base if x is a string.
2	long(x [,base]) Conv erts x to a long integer. base specifies the base if x is a string.
3	float(x) Conv erts x to a floating-point number.
4	complex(real [,imag]) Creates a complex number.
5	str(x) Conv erts object x to a string representation.
6	repr(x) Conv erts object x to an expression string.
7	ev al(str) Ev aluates a string and returns an object.
8	tuple(s) Conv erts s to a tuple.
9	list(s) Conv erts s to a list.
10	set(s) Conv erts s to a set.
11	dict(d) Creates a dictionary. d must be a sequence of (key,v alue) tuples.
12	frozenset(s) Conv erts s to a frozen set.
13	chr(x) Conv erts an integer to a character.
14	unichr(x) Conv erts an integer to a Unicode character.
15	ord(x) Conv erts a single character to its integer v alue.
16	hex(x) Conv erts an integer to a hexadecimal string.
17	oct(x) Conv erts an integer to an octal string.

MATHEMATICAL FUNCTIONS

1	abs(x)The absolute value of x: the (positive) distance between x and zero.
2	ceil(x)The ceiling of x: the smallest integer not less than x
3	cmp(x, y)-1 if x < y, 0 if x == y, or 1 if x > y
4	exp(x)The exponential of x: e^x
5	abs(x)The absolute value of x.
6	floor(x)The floor of x: the largest integer not greater than x
7	log(x)The natural logarithm of x, for x> 0
8	log10(x)The base-10 logarithm of x for x> 0.
9	max(x1, x2,...)The largest of its arguments: the value closest to positive infinity
10	min(x1, x2,...)The smallest of its arguments: the value closest to negative infinity
11	modf(x)The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
12	pow(x, y)The value of x**y.
13	round(x [,n])x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
14	sqrt(x)The square root of x for x > 0

RANDOM NUMBER FUNCTIONS

1	**choice(seq)A random item from a list, tuple, or string.**
2	randrange ([start,] stop [,step])A randomly selected element from range(start, stop, step)
3	random()A random float r, such that 0 is less than or equal to r and r is less than 1
4	seed([x])Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
5	shuffle(lst)Randomizes the items of a list in place. Returns None.
6	uniform(x, y)A random float r, such that x is less than or equal to r and r is less than y

TRIGONOMETRIC FUNCTIONS

1	**acos(x)Return the arc cosine of x, in radians.**
2	asin(x)Return the arc sine of x, in radians.
3	atan(x)Return the arc tangent of x, in radians.
4	atan2(y, x)Return atan(y / x), in radians.
5	cos(x)Return the cosine of x radians.
6	hypot(x, y)Return the Euclidean norm, sqrt(x*x + y*y).
7	sin(x)Return the sine of x radians.
8	tan(x)Return the tangent of x radians.
9	degrees(x)Converts angle x from radians to degrees.
10	radians(x)Converts angle x from degrees to radians.

MATHEMATICAL CONSTANTS

1	**pi** **The mathematical constant pi.**
2	e The mathematical constant e.

Date and time

- Epoch time is from january 1st 1970 until today in seconds.
- The module most commonly used is time.

Example

```
Import time
Print(time.time())
```

EXAMPLES CONTINUED

```
import time;

current_time = time.localtime(time.time())
Current_time2 = time.asctime( time.localtime(time.time()) )
```

- Can be also used to create a delay

```
Time.sleep(10)
```

Other time modules

- Datetime
- Calender
- Pytz
- dateutil

Conditional Statements

- Can be either true or false and based on the result we choose to perform an action that is suitable to the event.
- Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

Conditional statements

- If statement in python can be written as the follwing:

```
If x == 1:
    Print("correct")
```

- You can also add and else alone or with a second condition using elif (else if) as the following:

```
If x == 1:
    Print("correct")
Else:
    Print("Incorrect")
```

```
If x == 1:
    Print("correct")
Elif x == 2:
    Print("Incorrect")
```

Conditional statements

- You can add one or more condition in the same statement using and/or as the following:

```
X = 10
If x > 5 and x < 11:
    Print("perfect")
-----------------------------
If x > 10 or x < 30:
    Print("perfect")
```

Difference between if and elif

- Consider the following examples:

```
X = 100
If x == 100:
    Print("x is equal to 100")
If x > 17:
    Print("x is greater than 17")
```

```
X = 100
If x == 100:
    Print("x is equal to 100")
Elif x > 17:
    Print("x is greater than 17")
```

Nested conditions

- Example:

```
Car = 1
Color = "blue"
If car == 1:
    Print("user has a car")
    If Color == "blue":
        Print("Blue car found")
    Else:
        Print("Blue car not found")
Else:
    Print("User doesn't have a car")
```

Loops

- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times.

Loops

- Loops are used to repeat operations multiple times.
- Example:

```
For counter in range(5):
    Print("hello world")
```

- Side Note: in python you can use the following syntax to print multiple times:

```
Print("hello World\n" * 5)
```

Range function and for loops

- Returns an object that produces a sequence of integers from start (inclusive) to stop (exclusive).
- Syntax: range(Starting point, ending point, steps).
- If you didn't specify a starting point the sequence will start at zero.
- If you didn't specify the steps the sequence it will default to +1.

```
For I in range(0 , 10, 2):
    Print("test")
For I in range(10, 0, -2):
    Print("test")
```

While loops

- While loops is another type of loops where the loop keeps going until a condition is met.

```
X = 0
While x < 100:
    X += 1
    Print(x)
```

- While loops are used to create infinite loops in a system especially embedded systems.

LOOP CONTROL STATEMENTS

Break Statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
Continue Statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
Pass Statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

FILES

```
F = open("test.txt", "r")
f.read() # reads file , you can sepcify number of bytes too
f.tell() # shows pointer position
f.close()
---------------------------------------
F = open("test.txt", "w")
f.write("hello world")
f.close()
---------------------------------------
```

To avoid overwriting you can use append mode.

Functions

- Functions are way of bundling lines of code so that they may be re-used multiple times without having to write the same code again.
- You can send variables to functions.
- You can send integers, strings , sets , lists etc to a function.
- A function also returns data.

Functions

```
Def testfunction():
    Print("hello world")

Def sum(x, , y):
    Return x+y

Answer = sum(1,2)
Print(Answer)
```

Pass by reference vs value

- All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
def changelist( L ):
   L.append('a','b','c','d']);
   print ("Values inside the function: ", L)
   return

L= [1,2,3];
changelist( L );
print("Values outside the function: ", L)
```

Arguments

- There are many types of arguments for functions:
- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required arguments

```
Def test(x):
    Print(x)
    Return
```

Test(5) or Test()? Which is correct and why

Keyword argument

```
Def test(x):
    Print(x)
    Return

Test(x = 20)
```

- What if we have more than one argument?

Default argument

```
Def test(x = 20):
    Print(x)
    Return
```

Test(5) or Test()? Which is correct and why

VARIABLE-LENGTH ARGUMENTS

- This is used when you need to process more variables than you have set up in the first place , or you don't know how many variables you need exactly.

```
def test( x, *var ):
    print(x)
    for y in var:
        print y
    return;


test('yes')
test(1,2,3)
```

Scope of variables

- We have global variables and local variables.
- Local variables can be accessed only inside the function they are declared in while global variables can be accessed anywhere in the code.
- It is not good practice to use global variables a lot.
- If you need to do so make sure you have a lock function so that the valie doesn't change while another function is using the global variable.

EXAMPLE

```
GLOBVAR= 0
DEF SUM( X, Y ):
    GLOBVAR = X + Y
    PRINT("INSIDE : ", GLOBVAR)
    RETURN GLOBVAR;

# NOW YOU CAN CALL SUM FUNCTION
SUM( 10, 20 );
PRINT "OUTSIDE : ", GLOBVAR
```

Question

- What is the difference between input and raw_input?

Packages

- Python Packages are simply python files that contain python functions, classes, and variables. It's used to avoid having all python functions and classes in one file.

From testpackage import testfunction

FAMOUS PACKAGES

- OS
- SYS
- TIME
- NUMPY
- PANDAS
- MATPLOTLIB

Exceptions

- Exceptions is an abnormal behaviour that might cause the program to crash due to a certain event or error, therefore it is essential to have a way to deal with such errors or events.
- Python offers try and except as a way to deal with such incidents.
- you can raise exceptions in several ways by using the raise statement.
- You can also use finally to always execute a code

Regular expressions (regex)

- special sequence of characters that helps you filter and find other sub string or string or sets of strings, using a special filter with a unique syntax.
- We will import "re" to use this operation.
- We have several functions and methods that we can use to filter out and find what we desire.

MATCH METHOD

re.match(pattern, string, flags=0)

Sr.No.	Parameter & Description
1	pattern This is the regular expression to be matched.
2	string This is the string, which would be searched to match the pattern at the beginning of string.
3	flags You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

EXAMPLE

```
IMPORT RE
PATTERN = '^A...S$'
TEST_STRING = 'ABYSS'
RESULT = RE.MATCH(PATTERN, TEST_STRING)

IF RESULT:
  PRINT("SEARCH SUCCESSFUL.")
ELSE:
  PRINT("SEARCH UNSUCCESSFUL.")
```

- THE PATTERN IS: **ANY FIVE LETTER STRING STARTING WITH A AND ENDING WITH S**
-

METACHARACTERS

- To specify regular expressions, metacharacters are used. In the above example, ^ and $ are metacharacters.
- Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

[] . ^ $ * + ? {} () \ |

[] - SQUARE BRACKETS
SQUARE BRACKETS SPECIFIES A SET OF CHARACTERS YOU WISH TO MATCH.

HERE, [ABC] WILL MATCH IF THE STRING YOU ARE TRYING TO MATCH CONTAINS ANY OF THE A, B OR C.

Expression	String	Matched?
[abc]	a	1 match
	ac	2 matches
	Hey Jude	No match
	abc de ca	5 matches

CONTINUED

- You can also specify a range of characters using - inside square brackets.
- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].
- You can complement (invert) the character set by using caret ^ symbol at the start of a square-bracket.
- [^abc] means any character except A or B or C.
- [^0-9] means any non-digit character.

CONTINUED

- . - **PERIOD**
- A PERIOD MATCHES ANY SINGLE CHARACTER (EXCEPT NEWLINE '\N').

Expression	String	Matched?
..	a	No match
	ac	1 match
	acd	1 match
	acde	2 matches (contains 4 characters)

CONTINUED

- ^ - **CARET**
- THE CARET SYMBOL ^ IS USED TO CHECK IF A STRING **STARTS WITH** A CERTAIN CHARACTER.

Expression	String	Matched?
^a	a	1 match
	abc	1 match
	bac	No match
^ab	abc	1 match
	acb	No match (starts with a but not followed by b)

CONTINUED

- $ - **DOLLAR**
- THE DOLLAR SYMBOL $ IS USED TO CHECK IF A STRING **ENDS WITH** A CERTAIN CHARACTER.

Expression	String	Matched ?
a$	a	1 match
	formula	1 match
	cab	No match

CONTINUED

- * - **STAR**
- THE STAR SYMBOL * MATCHES **ZERO OR MORE OCCURRENCES** OF THE PATTERN LEFT TO IT.

Expression	String	Matched?
ma*n	mn	1 match
	man	1 match
	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

CONTINUED

- **+ - PLUS**
- THE PLUS SYMBOL + MATCHES **ONE OR MORE OCCURRENCES** OF THE PATTERN LEFT TO IT.

Expression	String	Matched?
ma+n	mn	No match (no a character)
	man	1 match
	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

CONTINUED

- ? - **QUESTION MARK**
- THE QUESTION MARK SYMBOL ? MATCHES **ZERO OR ONE OCCURRENCE** OF THE PATTERN LEFT TO IT.

Expression	String	Matched?
ma?n	mn	1 match
	man	1 match
	maaan	No match (more than one a character)
	main	No match (a is not followed by n)
	woman	1 match

CONTINUED

- **{} - BRACES**
- CONSIDER THIS CODE: {N,M}. THIS MEANS AT LEAST N, AND AT MOST M REPETITIONS OF THE PATTERN LEFT TO IT.

Expression	String	Matched?
a{2,3}	abc dat	No match
	abc daat	1 match (at daat)
	aabc daaat	2 matches (at aabc and daaat)
	aabc daaaat	2 matches (at aabc and daaaat)

- **| - Alternation**
- Vertical bar | is used for alternation (or operator).

Expression	String	Matched?
a|b	cde	No match
	ade	1 match (match at ade)
	acdbea	3 matches (at acdbea)

CONTINUED

- () - **GROUP**
- PARENTHESES () IS USED TO GROUP SUB-PATTERNS. FOR EXAMPLE, (A|B|C)XZ MATCH ANY STRING THAT MATCHES EITHER A OR B OR C FOLLOWED BY XZ

Expression	String	Matched?
(a|b|c)xz	ab xz	No match
	abxz	1 match (match at abxz)
	axz cabxz	2 matches (at axzbc cabxz)

CONTINUED

- \B - MATCHES IF THE SPECIFIED CHARACTERS ARE AT THE BEGINNING OR END OF A WORD.

Expression	String	Matched?
\bfoo	football	Match
	a football	Match
	afootball	No match
foo\b	the foo	Match
	the afoo test	Match
	the afootest	No match

CONTINUED

- \B - OPPOSITE OF \B. MATCHES IF THE SPECIFIED CHARACTERS ARE **NOT** AT THE BEGINNING OR END OF A WORD.

EXPRESSION	STRING	MATCHED?
\Bfoo	football	No match
	a football	No match
	afootball	Match
foo\B	the foo	No match
	the afoo test	No match
	the afootest	Match

CONTINUED

- \D - MATCHES ANY DECIMAL DIGIT. EQUIVALENT TO [0-9]

Expression	String	Matched?
\d	12abc3	3 matches (at 12abc3)
	Python	No match

CONTINUED

- \D - Matches any non-decimal digit. Equivalent to [^0-9]

Expression	String	Matched?
\D	1ab34"50	3 matches (at 1ab34"50)
	1345	No match

CONTINYED

- \s - Matches where a string contains any whitespace character. Equivalent to [\t\n\r\f\v].

Expression	String	Matched?
\s	Python RegEx	1 match
	PythonRegEx	No match

CONTIED

- \S - MATCHES WHERE A STRING CONTAINS ANY NON-WHITESPACE CHARACTER. EQUIVALENT TO [^ \T\N\R\F\V].

Expression	String	Matched?
\S	a b	2 matches (at a b)
		No match

CONTINUED

- \Z - MATCHES IF THE SPECIFIED CHARACTERS ARE AT THE END OF A STRING.

Expression	String	Matched?
Python\Z	I like Python	1 match
	I like Python Programming	No match
	Python is fun.	No match

EXAMPLES

```
# Program to extract numbers from a string

import re

string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'

result = re.findall(pattern, string)
print(result)

# Output: ['12', '89', '34']
```

EXAMPLES

- The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

```
import re


string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'


result = re.split(pattern, string)
print(result)
```

RE.SUB

- re.sub()
- The syntax of re.sub() is:
- re.sub(pattern, replace, string)
- The method returns a string where matched occurrences are replaced with the content of replace variable.

EXAMPLES

```
IMPORT RE

STRING = 'ABC 12\
DE 23 \N F45 6'

PATTERN = '\S+'

REPLACE = ''
NEW_STRING = RE.SUB(PATTERN, REPLACE, STRING)
PRINT(NEW_STRING)
```

RE.SEARCH()

- The `re.search()` method takes two arguments: a pattern and a string. The method looks for the first location where the RegEx pattern produces a match with the string.
- If the search is successful, `re.search()` returns a match object; if not, it returns `None`.
- `match = re.search(pattern, str)`

EXAMPLE

```
IMPORT RE

STRING = "PYTHON IS FUN"

MATCH = RE.SEARCH('\APYTHON', STRING)

IF MATCH:
  PRINT("PATTERN FOUND INSIDE THE STRING")
ELSE:
  PRINT("PATTERN NOT FOUND")
```

EXERCISES

- CHECK THAT A STRING CONTAINS NUMBERS FROM 1-4.
- MATCHA STRING THAT HAS A C FOLLOWED BY 2 T.
- LETTERS JOINED WITH A UNDERSCORE.
- FIND ALL THE WORDS BEGINNING WITH AN A.
- FIND ALL THE WORD ENDING WITH T.

Intro to Classes

- Classes as mentioned earlier are models or templates for a certain data type. Below is the syntax for creating our own data type as seen in the example below we created a data type rectangle (with width and height).

```
Class rectangle():
    Width = 0
    Height = 0
X = rectangle()
x.height = 7
x.width = 9
```