Python Introduction

Class 2

Today’s Agenda

- Version History
- Python 2 v/s Python 3
- Introduction To Predefined Functions And Modules
- How print() function works ?
- How To Remove Newline From print( ) ?
- Types Of Errors In Python
- Rules For Identifiers
- Python Reserved Words

Python Version History

- First released on Feb-20th -1991 (version 0.9.0)
- Python 1.0 launched in Jan-1994
- Python 2.0 launched in Oct-2000
- Python 3.0 launched in Dec-2008
- Python 2.7 launched in July 2010
- Python 3.6.5 launched on March-2018
- Python 3.7 launched on June-2018
- Python 3.8 launched on Oct - 2019
- Python 3.9 launched on Oct - 2020
- Python 3.10 launched on Oct - 2021
- Python 3.11 launched on Oct - 2022 [Latest version]

The Two Versions Of Python

- As you can observe from the previous slide , there are 2 major versions of Python , called **Python 2** and **Python 3**

- **Python 3** came in **2008** and **it is not backward compatible with Python 2**

- This means that a project which uses **Python 2** will not run on **Python 3.**

- This means that we have to **rewrite the entire project** to migrate it from **Python 2** to **Python 3**

Some Important Differences

- **In Python 2**

  ```
  print “Hello iNeuron”
  ```

- **In Python 3**

  ```
  print(“Hello iNeuron”)
  ```

- **In Python 2**

  ```
  5/2 →2
  5/2.0→2.5
  ```

- **In Python 3**

  ```
  5/2→ 2.5
  ```

- The way of accepting input has also changed and like this there are many changes

The Two Versions Of Python

- So to prevent this overhead of programmers , **PSF** decided to support **Python 2** also.

- But this support will only be till **Jan-1-2020**

- You can visit **https://pythonclock.org/** to see exactly how much time is left before Python 2 retires

Which Version Should I Use ?

- For beginners , it is a point of confusion as to **which Python version they should learn ?**
- The obvious answer is **Python 3**

Why Python 3 ?

- We should go with **Python 3** as it brings lot of new features and new tricks compared to **Python 2**

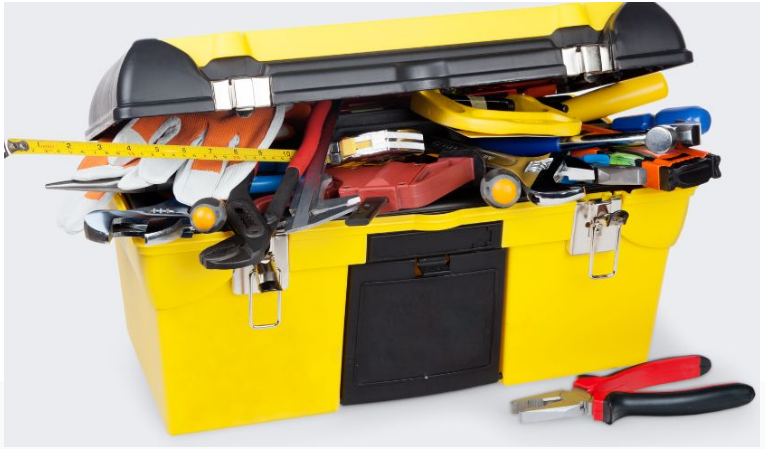
- **Moreover as per PSF,** *Python 2.x is legacy, Python 3.x is the present and future of the language*

- **All major future upgrades will be to Python 3 and , Python 2.7 will never move ahead to even Python 2.8**

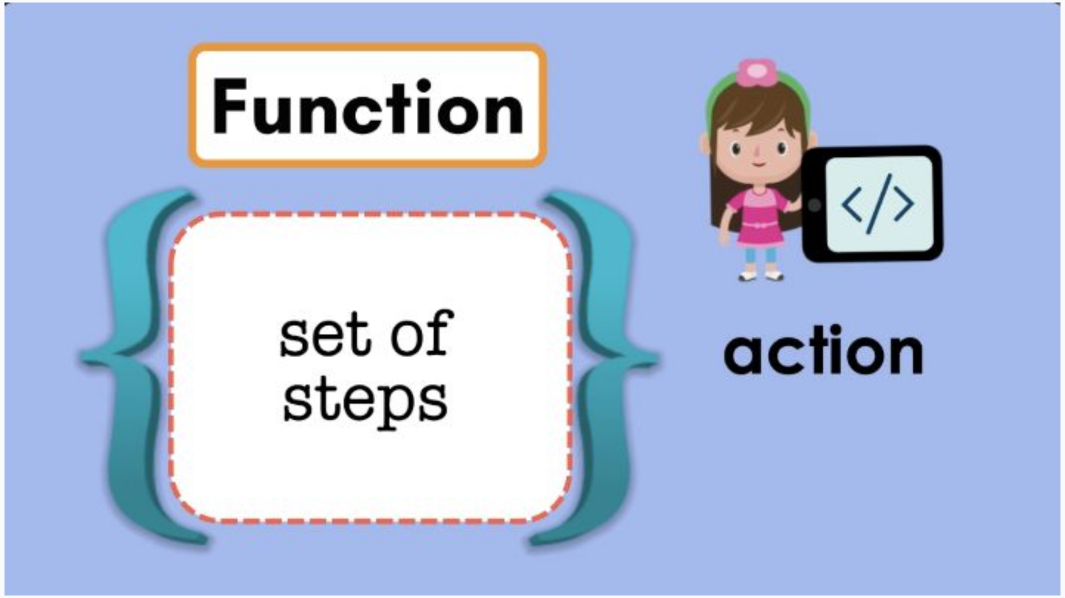
Types Of Predefined Function Provided By Python

- Python has a very rich set of predefined functions and they are broadly categorized to be of 2 types

  - Built In Functions
  - User defined functions



In computer programming, a function or subroutine is a **sequence of program instructions that performs a specific task**, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.



Built In Functions

- **Built in functions** are those functions which are always available for use .

- For example , **print()** is a **built-in function** which prints the given object to the standard output device (screen)

- As of version **3.6** , Python has **68** **built-in function** and their list can be obtained on the following URL :

  https://docs.python.org/3/library/functions.html

Built-in Functions in Python						
abs()	classmethod()	filter()	id()	max()	property()	str()
all()	compile()	float()	input()	memoryview()	range()	sum()
any()	complex()	format()	int()	min()	repr()	super()
ascii()	delattr()	frozenset()	isinstance()	next()	reversed()	tuple()
bin()	dict()	getattr()	issubclass()	object()	round()	type()
bool()	dir()	globals()	iter()	oct()	set()	vars()
bytearray()	divmod()	hasattr()	len()	open()	setattr()	zip()
bytes()	enumerate()	hash()	list()	ord()	slice()	__import__()
callable()	eval()	help()	locals()	pow()	sorted()	
chr()	exce()	hex()	map()	print()	staticmethod()	

What Is print( ) And How It Is Made Available To Our Program ?

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

How To Remove newline From print() ?

```
print("Hello User")

print("Python Rocks")
```

If we closely observe , we will see that the 2 messages are getting displayed on separate lines , even though we have not used any newline character.

This is because the function **print()** automatically appends a **newline character** after the message it is printing.

How To Remove newline From print() ?

If we do not want this then we can use the **print()** function as shown below:

```
print(“Hello User”, end=“”)
print(“Python Rocks”)
```

How To Remove newline From print() ?

The word **end** is called **keyword argument** in **Python** and it's default value is **“\n”**.

But we have changed it to **empty string**(“”) to tell **Python** not to produce any newline.

Similarly we can set it to **“\t”** to generate tab or **“\b”** to erase the previous character

Some Examples

1.

```
print("Hello User",end="\t")

print("Python Rocks")
```

2.

```
print("Hello User",end="\b")

print("Python Rocks")
```

Types Of Errors In Python

- Just like any other programming language , **Python** also has 2 kinds of errors:
  - **Syntax Error**
  - **Runtime Error**

Syntax Error

- Syntaxes are **RULES OF A LANGUAGE** and when we break these rules , the error which occurs is called **Syntax Error.**

- Examples of **Syntax Errors** are:
  - **Misspalled keywords.**
  - **Incorrect use of an operator.**
  - **Omitting parentheses in a function call.**

Runtime Errors (Exceptions)

- As the name says, Runtime Errors are errors which occur while the program is running.

- As soon as Python interpreter encounters them it halts the execution of the program and displays a message about the probable cause of the problem.

Runtime Errors (Exceptions)

- They usually occurs when interpreter counters a operation that is impossible to carry out and one such operation is **dividing a number by 0.**

- Since dividing a number by 0 is undefined , so ,when the interpreter encounters this operation it raises ZeroDivisionError as follows:

Functions Defined In Modules

A Module in Python is collection of functions and statements which provide some extra functionality as compared to built in functions.

We can assume it just like a header file of C/C++ language.

Python has 100s of built in Modules like math , sys , platform etc which prove to be very useful for a programmer

Variable

- Specific, case-sensitive name
- Call up value through variable name
- 1.79 m - 68.7 kg

```
height = 1.79
weight = 68.7
height
```

```
1.79
```

Calculate BMI

```python
height = 1.79
weight = 68.7
height
```

```
1.79
```

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

```python
68.7 / 1.79 ** 2
```

```
21.4413
```

```python
weight / height ** 2
```

```
21.4413
```

```python
bmi = weight / height ** 2
bmi
```

```
21.4413
```

Reproducibility

```python
height = 1.79
weight = 68.7
bmi = weight / height ** 2
print(bmi)
```

```
21.4413
```

Rules For Identifiers

- What is an identifier ?
  - Identifier is the name given to entities like **class**, **functions**, **variables** , **modules** and **any other object** in Python.

- Rules for identifiers:
  - Identifiers can be a combination of letters in **lowercase** (a to z) or **uppercase** (A to Z) or **digits** (0 to 9) or an **underscore** (_)
  - No special character except **underscore** is allowed in the name of a variable

Rules For Reserved Words

- **What is a Reserved Word?**
  - A word in a programming language which has a fixed meaning and cannot be redefined by the programmer or used as identifiers
- **How many reserved words are there in Python ?**
  - Python contains **33 reserved words** or **keywords**
  - The list is mentioned on the next slide
  - We can get this list by using **help()** in **Python Shell**

Rules For Reserved Words

These 33 keywords are:

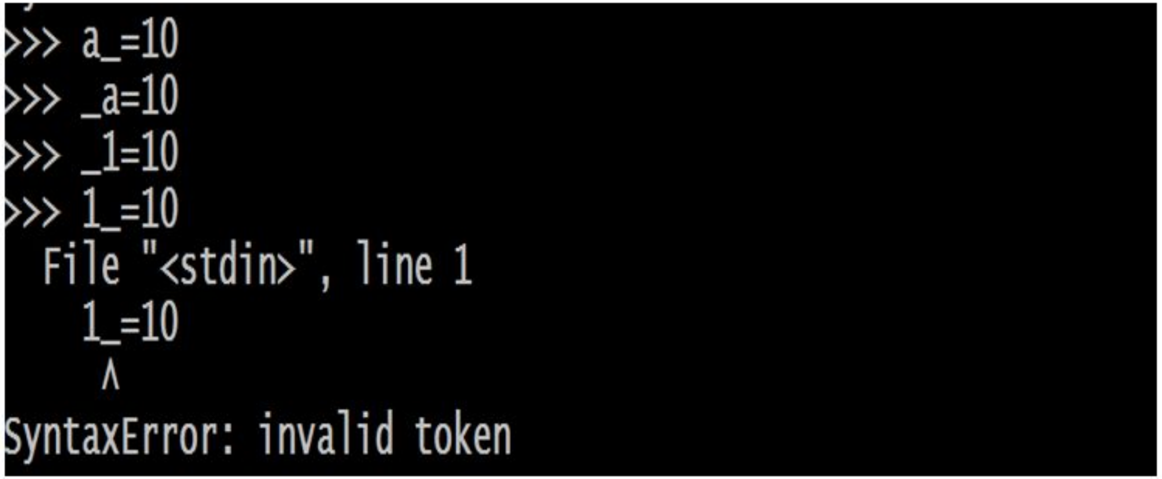
False , True , None ,def,
del ,import ,return , and , or ,
not ,if, else , elif , for , while ,
break , continue, is , as , in ,
global , nonlocal ,yield ,
try ,except , finally, raise,
lambda ,with ,assert ,class ,
from , pass

Some Important Observations:

1. Except False , True and None all the other keywords are in lowercase
2. We don't have else if in Python , rather it is elif
3. There are no switch and do-while statements in Python

Rules For Identifiers

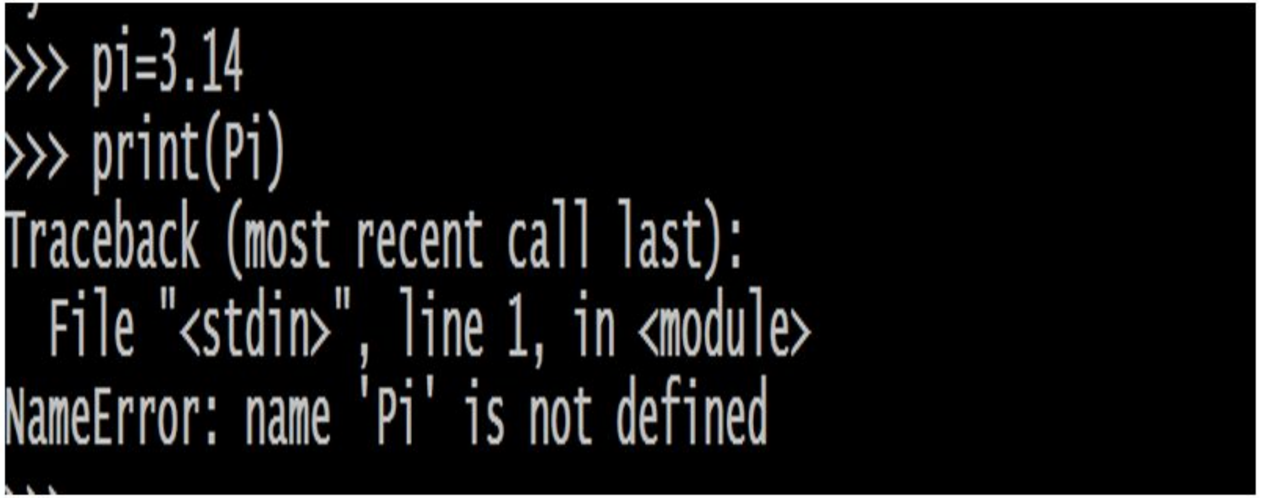
It must compulsorily begin with a underscore (_) or a letter and not with a digit . Although after the first letter we can have as many digits as we want. So **1a** is **invalid** , while **a1** or **_a** or **_1** is a **valid name** for an identifier.



```
>>> a_=10
>>> _a=10
>>> _1=10
>>> 1_=10
  File "<stdin>", line 1
    1_=10
     ^
SyntaxError: invalid token
```

Rules For Identifiers

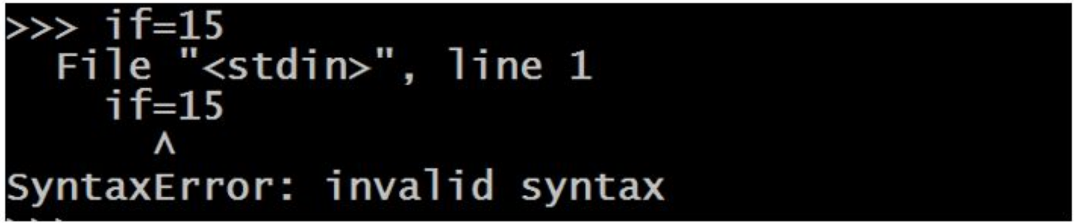
Identifiers are case sensitive , so **pi** and **Pi** are two different identifiers.



```
>>> pi=3.14
>>> print(Pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Pi' is not defined
```

Rules For Identifiers

- Keywords cannot be used as identifiers



```
>>> if=15
  File "<stdin>", line 1
    if=15
      ^
SyntaxError: invalid syntax
```

- Identifier can be of any length.