⇒ Python Course

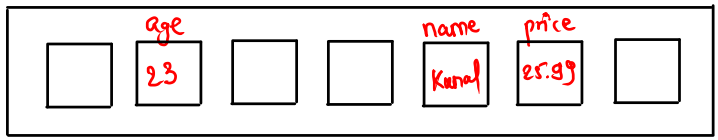
- Simple and easy
- Free and open source
- High level language
- Portable

first program

```
print ("Hello world")
```

Variables

```
name = "Kunal"
age = 23
price = 25.99
```



memory allocation

code

```
print ("my name is ", name);
```

here, we use comma (not +)

Note:- `print (type (nam))` // str

⇒ Data Type (Integers, String, Float, Boolean, None)

⇒ Keywords

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

Note :- Python is case sensitive

Type of Tokens :- Punctuator :-

(), {}, @, [], # etc.

Note:- Python is implicit typed lang. means we don't have to mention data type while declaring variables

⇒ Expression Execution

1) `A, B = 2, 3` declare in same line

2)
```
Txt = "@"
print ( 2 * Txt * 3)
```
@@@@@@

// str will get concatenated multiple time if multiply will numeric value

3)
```
A, B = "2", 3
Txt = "@"
print ( (A+Txt) * B)
```
2@2@2@

// with + str will be concatenated and then multiply B times

3) A, B = 2, 3

C = 4

print (A+B* C) // 14 (BODMAS)

4) A, B = 10 , 5.0

C = A* B

print (C) // 50.0 (float value)

5) Result of division operator with two integer will be float

A, B = 1, 2

C = A/B

print (C) 0.5

But, A, B = 1.5, 3

C = A // B 0.0 ← again got float value bcz we are dividing float and integer but now we have 0.0 bcz // is called as integer division

6) A, B = 12, 5

C = A // B

print (c) 2 (means # give floor integer value)

7) A, B = -12, 5

C = A // B -3 (always floor value means smaller)

8) A, B = 12, -5

C = A // B -3 (works as floor)

9) C = A % B

A, B = -5, 2 1

A, B = 5, 2 1

A, B = 5, -2 -1

we get -ve remainder only when denominator is -ve

(same as java)

means

num / denom	+	−	+	−
	+	−	−	+
	(+)	(+)	(−)	(+)

⇒ Type of operators

1) Airthematic (+, -, *, /, //, %, **)
2) Relational / Comparision (==, !=, >, <, >=, <=)
3) Assignment (=, -=, *=, /=, %=, //=, **=)
4) Logical (not, and, or)
5) Membership (in, not in)
6) Identity (is, is not)
7) Bitwise (&, |, ^)

i.e.,

⇒ (not True) and False or True

⇒ (False and False) or True

⇒ False or True ⇒ True

Ex:- operator precidence :- or > and > or

print (not True and False or True) True

⇒ Comments

single line comment

""" multiple line comment """

⇒ Input in Python

input() statement used to accept values from user

1) string input — `name = input("Kunal")` — when executed Kunal get printed and then accept for input

2) int input — `age = int(input("age : "))`

3) float input — `price = float(input("price : "))`

Input will always converted to str unless specifically mentioned

Note :- a **b mean a^b (valid airthematic operator)

⇒ Conditional statement

if - elif - else (SYNTAX)

```
if ( condition ) :
        Statement 1
elif ( condition ) :
        Statement 2
else :
        Statement 3
```

colon is important

Ex:-

```
light = input("light : ")
if(light == "red"):
    print("stop")
elif(light == "yellow"):
    print("look")
elif(light == "green"):
    print("go")
else:
    print("light is broken")
```

⇒ Single line if / Ternary operator

food = input("food : ")

↳ cat = "Yes" if food == "cake" else "No"

↳ print("sweet") if food == "cake" or food == "jalebi" else print("not sweet")

Note:- clever if / Ternary operator

```
age = int(input("age:"))
vote = ("yes","no")[age >= 18]
```

⇒ Type Conversion (implicitely changing) and Type casting (explicitely changing)

Ex:- 1)
```
a = 2
b = 5.2
sum = a+b      7.2   (float is superiar)
```

2)
```
a = "2"
b = 5.2
sum = a+b      // error  (not possible implicitely)
```

So explicity
```
a = int("2")      (explicitly)      or      a = str(2)
b = 5.2
sum = a+b
```

⇒ String

1) concatenation "hello" + "world"

2) len(str) // provide length of str

code

```
str = "This is \t a \n sentence"
```

Note:- Python has 0-based indexing

```
str = "kunal_swi"
      0 1 2 3 4 5 6 7 8 9

print(str[2])    #n
```

Note:- str is immutable in here as well

→ Slicing (accessing part of str)

syntex str[si : ei+1]

```
str = "kunalswi"
      0 1 2 3 4 5 6 7 8

print(str[1:4])  #una

print(str[5:])   # swi

print(str[:3])   # kun
```

Python automatically fills si or ei if missed

-ve index

```
str = "Apple"
      -5 -4 -3 -2 -1

print(str[-3:-1])  # pl
```

↳ not included

⇒ Some functions to remember

str = " i am a docter. "

1) str.endsWith(" er. ")

return true

2) str = str.capitalize ()

Capitalize first character

3) str.replace (old , new)

replace all occurences of old substr to new substr

4) str.find (word)

return first index of first occurence of word and -1 if word is not present

5) str.count (" am ")

count of " am " in str

⇒ List (same as array or arraylist)

Ex:- marks = [1 , 2 , 3 , 4 , 5] # marks[1] gives 2

student = ["Karen" , 85 , "Delhi"] valid

Note :- str is immutable and list is mutable (same as java)

print(len (student)) # 3

Note :- list sliing is also possible

output in console

marks [1 : 4] # [1 , 2 , 3]

apply same rules as string sliing

Some inbuilt functions of list

```
list = [2, 1, 3]
list.append(4)          # [2, 1, 3, 4]
list.sort()             # [1, 2, 3, 4]
list.sort(reverse = True)   # [4, 3, 2, 1]
```

Note :- sort and append doesnt return anything ☆

```
list.reverse()          # [4, 3, 1, 2]
list.insert(idx, ele)
            (1, 5)      # [2, 5, 3, 4]
list.remove(1)          # remove first occ. of 1
list.pop(idx)           # remove ele. at idx
```

⇒ Tuples (similar to list)

↳ immutable list

tup = (2 , 1 , 3 , 2) means tup[0] gives 2

but tup[1] = 5 # error

Note:- slicing in tupple work same as list

→ Inbuilt functions

tup.index(ele) # return idx of first occ. of ele

tup.count(ele) # return count of ele

Note:- we can print list are tupple using print (without loops)

⇒ Dictionary (same as hashmap)

↳ unaltered, mutable and don't allow duplicate key

Syntax

```
dict = {
        "name" : "kunal",
        "cgpa" : 9.6,
        "marks" : [ 98, 97, 95 ],
        12.99 : 94.4
}
```

also can be printed with print (without loops)

Note:-

null_dict = { }

```
dict["name"]          # accessing values
dict["name"] = "Suri"  # updating values
```

→ Nested dictionary

```
student = {
        "name" : "kunal"
        "subjects" : {
                "phy" : 97
                "chem" : 98
        }
}
```

accessing

```
student ["subjects"] ["chem"]
# 98
```

→ Inbuilt functions

```
myDict.keys( ) #returns all keys

myDict.values( ) #returns all values

myDict.items( ) #returns all (key, val) pairs as tuples

myDict.get( "key" ) #returns the key according to value

myDict.update( newDict ) #inserts the specified items to the diction
```

Note:-

wrong key

```
print( student ["name2"] )
# error

print ( student . get ("name2"))
# no-error
  returns None
```

```
new_dict = { "city" : "Delhi", "age" : 26 }
student.update (new_dict)
```

⇒ Set in Python (same as hashset)

↳ unordered, immutable & unique

syntax

```
nums = { 1, 2, 3, 2 }    # stored as { 1, 2, 3 }
```

Note:- we can't add list and dict in set bez they are mutable

we can only store int, boolean, float, str, tuple

Empty set

```
nums = { }      # not allowed
                  bcz this is dictionary
```

✗

So

```
nums = set ()   # correct way
```

✓

→ Inbuilt methods

```
set.add( el )  #adds an element

set.remove( el )  #removes the elem an

set.clear( )  #empties the set

set.pop( )  #removes a random value
```

Note :-

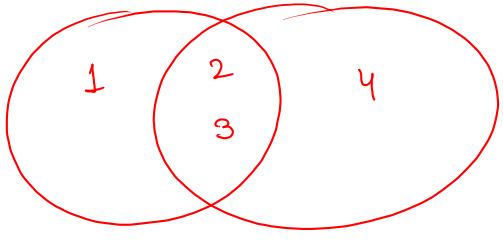
set is mutable

but elements in set is immutable

```
set.union( set2 )  #combines both set values & returns new

set.intersection( set2 )  #combines common values & returns new
```

Ex :-



set1 = { 1, 2, 3 }

set2 = { 2, 3, 4 }

```
print ( set1.union(set2))          # {1, 2, 3, 4}
print ( set1. intersection ( set2))  # { 2, 3 }
```

⇒ Loops

Syntex

while

```
i = 1
while i <= 5 :
    print ("Hi")
    i += 1
```

for

```
list = [1, 2, 3]
for el in list :
    print (el)
```

for loop with else (optional)

```
for el in list :
    print ("Hi")
else :
    print ("END")
```

will be executed once loop has end

→ range()

↳ returns the sequence of numbers, starting from 0 by default, and increments by 1 by default and stops before a specified number

syntax

```
range(start?, stop, step?)
```

code

```
for el in range(5):          # 0 1 2 3 4
    print(el)

for d in range(1,5):         # 1 2 3 4
    print(el)

for el in range(1,5,2):      # 1 3 5
    print(el)
```

→ Pass statement

↳ pass is used to do nothing. It is used as a placeholder for future code.

Code

```
for i in range(10):
    pass
```

⇒ Functions in Python

syntax

```
def fun_name(para1, para2, .....):   # function definition
    # some work
    return val

fun_name(arg1, arg2, .....)          # function calling
```

Note:- function is not returning anything

will provide back None

↳ same as null

Note:- Seperator is automatically defined as space

⇒ Recursion

```
def show(n):
    if(n == 0):
        return
    print(n)
    show(n-1)
```

```
def fact(n):
    if(n==0 or n==1):
        return 1
    else
        return n * fact(n-1)
```

⇒ File I/O

↳ reading and writing data

open a file

```
f = open("file_name", "mode")
```

"file_name" → sample.txt

"mode" → r : read mode
w : write mode

```
data = f.read()
f.close()
```

different modes

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)