Beginner

09 April 2025 11:49

Input and Output:

- print always has brackets in it
- input has print() function inside it and waits till an input is given
- input().split() uses spaces to find different values
- Tab spaces used to mark block
- if , elif , else:
- age = int(age_input) """ used to change input to integer"""

```
a = "Hello World"
b = 10
c = 11.22
d = ("Geeks", "for", "Geeks")
e = ["Geeks", "for", "Geeks"]
f = {"Geeks": 1, "for":2, "Geeks":3}


print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
print(type(f))
```

used to get the type of the data types

Formating:

```
amount = 150.75
print ("Amount=${:.1f}".format(amount))
```

round-off the value to 150.8

Formating using f strings:

```
name = "Charlie"
age = 28
formatted_string = f"Name: {name}, Age: {age}"
print(formatted_string)
```

FreeCodeCamp:

- Rock paper scissor game::

```
import random

def get_choices():
    player_choice = input("Enter the choice (rock, paper, scissors):")
    options = [ "rock" , "scissors" , "paper" ]
    computer_choice = random.choice(options)

                                    #leaving a line does not affect the code
    choices = { "player" : player_choice , "computer" : computer_choice}
    return choices
```

```
def check_win(player, computer):
    print (" you chose:" + player + ", computer chose:" + computer )
    print (f"you chose: {player} , computer chose: {computer}")
    if player == computer:
        return "It's a tie"
    elif player == "rock" :
        if computer == "scissors":
            return "Rock smashes scissors! You win "
        else:
            return "Paper covers rock! You lose"
    elif player == "scissors" :
        if computer == "rock":
            return "Rock smashes scissors! You lose "
        else:
            return "Scissor cut paper! You win"
    esle :
        if computer == "scissors":
            return "Scissor cut paper! You lose "
        else:
            return "Paper covers rock! You win"
choices = get_choices()
p_choice = choices[player]
c_choice = choices[computer]
result = check_win(p_choice, c_choise)
print(result)


def greeting():
    return "Hi"                  #no need of symbols to mark the end

response = greeting()
print(response)

choices = get_choices()
print(choices)

#dict = { "name" : "flower" , "colour" : "red" }
#choices_out = get_choices()

food = ["pizza", "carrot", "eggs" ]
dinner = random.choice(food)


a=3
b=5
if a>b:      # == is a check operator !=
    print ("yes")

age = 25
print (f"my age is {age} years old") # only f-string has this option
if age> 18 :
    print ("you are a adult")
elif age>12:
    print ("you are a child")
```

```
"""print ("Hello World")

name = input("Enter your name:") # get the input and waits till the user enters the
output

print (name)"""


'''

# should not use python keywords for identifiers

# like : if="Madhu"

#Expression:

# ";" is used to run multiple lines of code in the same line

name = "Madhu" ;print(name)# indentation do matter!

print(type(name)== str)

print(isinstance(name, str))

age = 2 ;

print (isinstance(age, int))# python defines the type automatically

#force

age = float (2)

#age = int("what is this going to produce")

## complex, bool, list, dict, set, tuple are the types in python

#Operators:

# + - * //- floor division cuts of the floating value %: reminder

print("Cooper"+" is a good dog")

age =8

age +=8 #age=age+8

print (age)

#Comparision Operators:

a=1

b=2

print(a<=b) #less than or equal to sign#False or True

condition1 = True
```

```python
not condition1

#or shows the last value if all the previous values are false without checking the last value

# [] , False , 0 represent false value

print([] and False)

print(8<<1)

#'is' used to equate objects

#'in' find elements in list

# Ternary operator

def is_adult(age):

    return True if age>18 else False

print(is_adult(20))

name = "Madhu"

pharse = name + " is my name"

name += " is my name"

print("""Madhu is 22

years old"""

)

#String Method:

print("madhu".upper())# .lower() islower(): checks for is all letters in the string is lower

#isalnum()

#isdecimal()

#lower()

#title : makes all the words first letter capital

#startswith() : check if the string starts with a specific substring

#endswith():

#replace(): to replace a part of the string

#strip() : trim the white spaces
```

```
#join() : to append new letters to a string

#find() : to find the position of a substring

# these does not change the actual value of the variables

print(len(name))

print("au" in name)

print ("mad" in name)

Quotes = " madh\"au"#or use '' for the string or "" for \' #escape character

#\n for new line

print (Quotes)

print (name[1]) #name [1:2] for the range and only letter excluding index 2

#starts with 0 and -1 from end

done = True

if done:

   print ("yes")

else:

   print ("no")

#numbers all are True except for 0(also empty strings) including negative values
and

print(type(done) == bool)

#complex numbers: 'j'

complex = 2+3j # num = complex(2,3)

#print (num.real,num.imag) # displayed as floats

#abs(-5.5) will give 5.5

print(round(5.49,1)) # 5.5

#Enum

from enum import Enum

class state(Enum):

   INACTIVE = 0

   ACTIVE = 1
```

```
print(state.ACTIVE.value) #1 just state.ACTIVE results in state.Active only!

#state.ACTIVE or state['ACTIVE'] OR state(1) all result in same state.ACTIVE

print(len(state))#2

print("What is your age")

age = input()  # waits for ENTER key

print ("Your age is " + age)


#List:

items = ["dog" , "cat", 1, True]

print("dog" in items) # True

print(items[1]) #cat

items[1] = "human"

print(items) # will update the list

items[-3] = "cat"# will revert the list

#slicing: items[:3] will give till value 1

items.append("last_element")

print(items)

items.append("Table")

items.extend(["Chair","sofa"]) # is same as += [] or will add char by char

print(items)

items.remove("Table") #if not prensent throws error

print(items.pop()) #prints the last element for the last time

print(items)

items.insert(2,"Car") #inserts at 2nd position

items [1:1] = ["Test1", "Test2"] #adds to the 1st position

#items.sort() #only numeric or alphabet

print(items)# b > B so give items.sort(key=str.lower)

#Copying a list without link

itemscopy = items[:]# also print(sorted(items, key= str.lower)) does not change
the actual value of list
```

```
#TUPLES:

names = ("Madhu" , "Pranes", "Vasee", "Rishi", "Vatson")

#names[-1]

#names.index("Madhu")

print(len(names))

print("Pranes" in names)

print(sorted(names)) #is possible as tuples can't be modified to sort tuple

#Can be concatinated new_names = names + ("Mithan") ,but can't changed : Tuples

#Dictionary::

home = {"name" : "Madhu", "age" : 20, "what" : "next","car": "bmw"}

print(home.get("name")) # gets value Madhu same as home["name"] but wont show error if the key value is not present

#get("colour", "brown") prints brown as home does not contain colour key

print(home.pop("name"))

print(home.popitem())# pop last item

print(list(home.keys()))

print(home)

print("age" in home) # True

print(list(home.items())) # tuple values inside a list

print(len(home))

home["Bike"] = "yamaha"#adds new value to dictionary

del home["Bike"]

new_home = home.copy() # duplicates


#Sets::

#values can't duplicate

car= {"auto", "bmw", "audi"}

bike = { "yamaha", "bmw"}

# &: for intersect, | : for union, - : for subtracting the elements, len(), >< for subset
```

```
print(car & bike)

#Functions::

def hello(name):

   print("hello " + name)

hello("Madhu")

def hi(value_entered):

   value_entered = 2

a=7

hi(a)

print(a)

#values of dictionary do change inside the function whereas the value of 'a' does
not change

# multiple values are returned using , which return as tuples from the function


#scope of function::

age = 3

def display():

   print(age)# age can also be accessed from here

print(age)

display()

#nested function::

def phrase(sent):

   def say(word):

      print (word)

   words = sent.split(' ')

   for word in words:

      say(word)

phrase("I like black colour")

#nonlocal::
```

```
def cal():
    count=0
    def calculate():
        nonlocal count
        count=count+1
        print(count)
    calculate()
cal()
#Closure::
def talk():
    count=0
    def speak():
        nonlocal count
        count+=1
        return count
    return speak
call=talk()
print (call())
print (call())

#Objects::
age =8
print (age.real)
print (age.imag)
print (age.bit_length())
items = [1, 2, 3]
items.append(4)
items.pop()
print (id(items))
```

#dictionarys are mutable while integer are immutable i.e They create a new object when incremented

```
#Loops::

count=0

while count<=10:

    print ("Madhu")

    count+=1

items= [1,2,3,4]

for index, item in enumerate(items):

    print (index, item)

for i in range(3,10):

    if i==5:

        continue#break will exit the loop and does not execute the line below them

    print (i)


#Class and Inheritance::

class Animal:

    def walk(self):

        print("walking...")

class Dog(Animal):

    def __init__(self,name,age):

        self.name =name

        self.age = age

    def bark(self):

        print ("bark")

madhu = Dog("mithan",8)

print (madhu.name)

print (madhu.age)

madhu.bark()

madhu.walk()
```

```
#Modules::

import dog

dog.bark()# can give bark() if :"from dog import bark" is given

#math for math utilities

#re for regular expressions

#json for work with Json

#datatime to work with date and time

#sqlite3 to use SQLite

#os for Operating System utilities

#random for random number generation

#statistics

#requests for HTTP network requests

#http to create HTTP servers

#urllib to manage URLs

#Arguments from the shell

import sys

name = sys.argv[1]

print ("Hello "+ name)

import argparse

parser = argparse.ArgumentParser( description = "This shows the name of the
colour entered ")

parser.add_argument("-c","--red", metavar = "colour" , required = True, help =
"the colour to display")

#add "choices = {'red', 'green'}" after required will make a constrain

args = parser.parse_args()

print(args.red)


#Lambda function::

multiply = lambda a,b : a*b

print (multiply (3,4))
```

```
#map():: maps items in a list to the input of functions

items = [ 1, 2, 3, 4, 5, 6]

mul = map (lambda a:a*2, items)

print (list(mul))

#filter():: maps the same and displays those which are True

result = filter(lambda a: a%2 == 0 , items)

print (list(result))

#reduce():: for iterative values where we have to store the value from previous
items

from functools import reduce

car= [("bmw", 32), ("auto", 45)]

sum = reduce(lambda a , b: a[1] + b[1] , car)

print(sum)

#Recursion::

def factorial(num):

   if num==1:return 1

   return num*factorial(num-1)

print(factorial(5))


#Decorators:: when we need to change the working the function but not the
function itself

def logtime(func):

   def wrapper():

      print ("Before")

      val = func()

      print ("After")

      return val

   return wrapper

@logtime

def hello():
    print("Hello")

hello()

#print(help(<class_name>)) will print the description plus the docstring of the
module

#Annotations:: ignored by python ide but can be integrated if needed for static
type errors before running the module

def increment (n : int) -> int:

    return n+1

count:int = "mad" # to check the value is integer or not but python this ide does
not check for correctness

print(increment(4))

#Exceptions::

#try:<block>

#except <error_type>:<block>

#except <error_type>:<block>

#except: <covers_all_types>

#else: <executes if no error is encountered>

#always: < gonna be executed everytime>

try:

    result = 2/0

except ZeroDivisionError:

    print("an Error occured")

finally:

    result=1

print(result)

raise Exception("What the f*ck is going on")#used to raise error if needed


#exception inside class::

class DogNotFoundException(Exception):

    print ("inside")# prints this statement
    pass# used if class dont have methods or function dont have code inside them

try:

    raise DogNotFoundException()

except DogNotFoundException:

    print("oh!! the dog is lost")

#How to open up files and read its content?

filename = "/user/<path to the directory"

with open (filename, "r") as file :

    content = file.read()

    print(content)

# pip install <packages_we_need> : to download packages from the shell


#List Compressions::

numbers = [1, 2, 3, 4, 5]

PowersOf2 = [n**2 for n in numbers]

print (PowersOf2)


#Polymorphism::

class Dog:

    def eat(self):

        print("Eating dog food")

class Cat:

    def eat(self):

        print("Eating cat food")

animal1= Dog()

animal2= Cat()

animal1.eat()

animal2.eat()

#Output:

#Eating dog food
```

```
#Eating cat food

#How to compare different class when called?


class Dog:

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def __gt__(self, other):#other methods are there for logical and arthmetic 
operations to perform

        return True if self.age > other.age else False

roger = Dog("Roger", 4)

syd = Dog ("Syd", 7)

print(syd > roger)

#Output:

#True

'''


#Blackjack Project::


import random

#suit = suits[0]

#rank = "K"

#value = 10

#print ("Your card is : " + rank + " of " + suit )

#suits.append("snakes")

#rank = card[1]

#if rank== "A":

#    value = 11

#elif rank == "K" or rank == "J" or rank == "Q":

#    value = 10
```

```
#else:
#    value = rank
#rank_dict = {"rank" : rank , "value" : value}
#print(rank_dict["rank"] , rank_dict["value"])
class NamingCards:
    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank
    def __str__(self):
        return f"{self.rank['rank']} of {self.suit}"
class Deck:
    def __init__(self):#creates cards deck with ordered sets of shape and number
        suits = ["hearts","clubs", "spade", "diamonds"]
        ranks = ["A", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
        rank_list = []
        self.cards = []
        for rank in ranks:
            if rank== "A":
                value = 11
            elif rank == "K" or rank == "J" or rank == "Q":
                value = 10
            else:
                value = rank
            rank_list.append({'rank':rank, 'value': value})
        for suit in suits:
            for rank in rank_list:
                self.cards.append(NamingCards(suit,rank))
    def shuffle(self):#shuffles the ordered pair of cards
        random.shuffle(self.cards)
    def deal(self, number):#returns some number of last cards from deck
        CardsDelt = []
        for i in range(number):
            if len(self.cards)>0:
                CardsDelt.append(self.cards.pop())
        return CardsDelt
class Hand:
    def __init__(self, dealer = False):
        self.cards = []
        self.value = 0
        self.dealer = dealer
    def add_card(self, CardsDelt):
        self.cards.extend(CardsDelt)
        print(self.cards)
    def calculate_value(self):
        self.value = 0
        has_ace = False
        for card in self.cards:
            card_value = int(card.rank["value"])
            self.value += card_value
            if card.rank["rank"] == "A":
                has_ace = True
        if has_ace and self.value > 21:
            self.value -=10
    def get_value(self):
        self.calculate_value()
        return self.value
    def is_blackjack(self):
        return self.get_value() == 21

    def display (self):

        print(f'''{"Dealer's" if self.dealer else "Your"} hand:''')

        for card in self.cards:

            print(card)

        if not self.dealer:

            print("Value:", self.get_value())

        print()


Deck1 = Deck()

Deck1.shuffle()

hand = Hand()

hand.add_card(Deck1.deal(2))

print(hand.display())
```