Day - 21

**Snake Game: Part 2**

**Inheritance & List Slicing**

Inheritance & Slicing, Snake: Food-collision, Wall-collision, Tail-collision, Score & Game-over

**21.1 Inheritance in python**

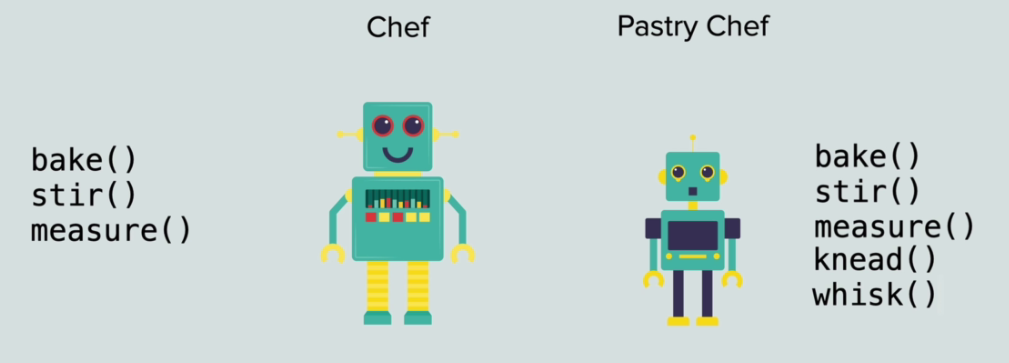
To declare a child-class we need to load the ***\_\_init()\_\_*** of the parent-class and pass the parent-class name as a parameter of child-class.

**class** child\_class\_name(parent\_class\_name):

**def** \_\_init\_\_():

super().\_\_init\_\_()

* Inside the ***\_\_init\_\_()*** of child we call the of parent ***super().\_\_init\_\_()***.



**21.2 Accessing and Modifying parent-class properties and methods**

|  |  |
| --- | --- |
| * Accessing parent-class properties and methods:   #*parent class*  **class** Animal:  **def** **\_\_init\_\_**(self):  **self.**num\_eyes = 2  **def** **brethe**(self):  **print**("Inhale Exhale.")  #*child class*  **class** Fish(Animal):  **def** **\_\_init\_\_**(self):  **super**()**.\_\_init\_\_**() #*call \_\_init\_\_() of parent-class*    **def** **swim**(self):  **print**("Moving in the water")  #*Creating object of child-class.*  #*Also contain the properties & methods of parent class*  nemo = **Fish**()  nemo**.swim**()  #*Accessing parent-class properties and methods*  **print**("No. of eyes : ", nemo**.**num\_eyes)  nemo**.brethe**()  #*python inheritance\_demo.py* | * Modifying parent-class methods: ff   #*parent class*  **class** Animal:  **def** **\_\_init\_\_**(self):  **self.**num\_eyes = 2  **def** **brethe**(self):  **print**("Inhale Exhale.")  #*child class*  **class** Fish(Animal):  **def** **\_\_init\_\_**(self):  **super**()**.\_\_init\_\_**() #*call \_\_init\_\_() of parent-class*  **def** **swim**(self):  **print**("Moving in the water")      #*modifying parent-class's method*  **def** **brethe**(self):          #*loading parent's method*  **super**()**.brethe**()  **print**("Doing this underwater.") |

**21.3 Detect Collisions with Food**

|  |  |
| --- | --- |
| **import** turtle  **import** random  **class** Food(turtle**.**Turtle):  **def** **\_\_init\_\_**(self):  **super**()**.\_\_init\_\_**()  **self.shape**("circle")  **self.color**("red")  **self.turtlesize**(0.4, 0.4)  **self.penup**()  **self.speed**("fastest")  **self.refresh**()    **def** **refresh**(self):          random\_x = random**.randint**(-380, 380)          random\_y = random**.randint**(-330, 330)  **self.goto**(random\_x, random\_y)  #*python food.py* | **. . .**  **import** food  **. . . .**  #*create a snake object*  snAKe = snake**.Snake**()  foOd = food**.Food**()  brDr = border**.Border**()  #*update the screen after loading the whole snake*  scr**.update**()  . . . . .  **while** game\_on:      scr**.update**()      time**.sleep**(0.2)        snAKe**.move**()      #*Collition with food*  **if** (snAKe**.**head**.distance**(foOd)) **<=** 10:  **print**("noom noom !!!")          foOd**.refresh**()      #*snake[0].left(90)*      #*Stop the snake at boundary.*  **if** (snAKe**.**head**.xcor**() **>=** 390) **or** (snAKe**.**head**.xcor**() **<=** -390) **or** (snAKe**.**head**.ycor**() **>=** 340) **or** (snAKe**.**head**.ycor**() **<=** -340):          game\_on = **False**  #*python snake\_main.py* |

**21.4 Create a Scoreboard and Keep Score**

|  |  |
| --- | --- |
| **import** turtle  ALIGNMENT = 'center'  FONT = ('Arial', 16, 'normal')  **class** Scoreboard(turtle**.**Turtle):  **def** **\_\_init\_\_**(self):  **super**()**.\_\_init\_\_**()  **self.hideturtle**()  **self.color**("#3c412c")  **self.penup**()  **self.speed**("fastest")  **self.goto**(0, 310)  **self.**score = 0  **self.update\_scoreboard**()  **def** **gameover**(self):  **self.goto**(0 , 0)  **self.write**(f"GAME OVER", move= **False**, align= ALIGNMENT, font= FONT)  **def** **update\_scoreboard**(self):  **self.write**(f"Score : {**self.**score}", move= **False**, align= ALIGNMENT, font= FONT)  **def** **icrease\_score**(self):  **self.**score += 1  **self.clear**()  **self.update\_scoreboard**()    #*python scoreboard.py* | **import** scoreboard  scrBrd = scoreboard**.Scoreboard**()  **if** (snAKe**.**head**.distance**(foOd)) **<=** 10:          score += 1  **print**("noom noom !!!")          foOd**.refresh**()          scrBrd**.icrease\_score**()  #*python snake\_main.py* |

**21.5 Detect Collisions with the Wall**

    #*Wall Collision: Stop the snake at boundary.*

**if** (snAKe**.**head**.xcor**() **>=** 360) **or** (snAKe**.**head**.xcor**() **<=** -360) **or** (snAKe**.**head**.ycor**() **>=** 310) **or** (snAKe**.**head**.ycor**() **<=** -310):

        scrBrd**.gameover**()

        game\_on = **False**

*# inside the while loop*

#*python snake\_main.py*

**21.6 Detect Collisions with your own Tail**

We do this in our Snake class in ***snake.py.*** First we extend the body as below. By redefining and adding ***extend\_body()*** and ***add\_segments()***

**import** turtle

STARTING\_POSITION = [(10, 0), (0, 0), (-10, 0)]

MOVE\_DISTANCE = 10

UP = 90

DOWN = 270

LEFT = 180

RIGHT = 0

**class** Snake:

**def** **\_\_init\_\_**(self):

**self.**snake = []

**self.create\_sanke**()

**self.**head = **self.**snake[0] #*This line is called after create\_sanke(), unless error occurs*

    #*create\_sanke(self), add\_segments(self, position) creates the snake body first*

**def** **create\_sanke**(self):

**for** pos **in** STARTING\_POSITION:

**self.add\_segments**(pos)

**def** **add\_segments**(self, position):

            sqrs = turtle**.Turtle**(shape = "square")

            sqrs**.shapesize**(0.4, 0.4, 0)

            sqrs**.color**("#3c412c")

            sqrs**.penup**()

            sqrs**.goto**(position)

                #*By default turtle size is 20 X 20. So ours will be 8 X 8. Distance from each other will be 10.*

**self.**snake**.append**(sqrs)

    #*Extend the body when collide with food*

**def** **extend\_body**(self):

        #*"-1" is used to indicate the "last" element of the list*

**self.add\_segments**(**self.**snake[-1]**.position**())

**def** **move**(self):

        #*Reverse for loop. Move following/backward segments to forward.*

**for** i **in** **range**(len(**self.**snake)-1, 0, -1):

            new\_x = **self.**snake[i-1]**.xcor**()

            new\_y = **self.**snake[i-1]**.ycor**()

**self.**snake[i]**.goto**(new\_x, new\_y)

            #*Shows one by one move "scr.update()". set higher value with higher "delay" to understand. Shows one by one move slowly: time.sleep(1).*

        #*moving the first segment*

**self.**head**.forward**(MOVE\_DISTANCE)

**def** **up**(self):

**if** **self.**head**.heading**() **!=** DOWN:

**self.**head**.setheading**(UP)

**def** **down**(self):

**if** **self.**head**.heading**() **!=** UP:

**self.**head**.setheading**(DOWN)

**def** **left**(self):

**if** **self.**head**.heading**() **!=** RIGHT:

**self.**head**.setheading**(LEFT)

**def** **right**(self):

**if** **self.**head**.heading**() **!=** LEFT:

**self.**head**.setheading**(RIGHT)

#*python snake.py*

* Detect collision with tail:

**while** game\_on:

    #*moving the snake segments. first it will move like a Caterpillar. Need to use "update" and disable "tracer". Update screen here when all snake is moved*

    scr**.update**()

    time**.sleep**(0.1)

. . . . .

    #*Wall Collision: Stop the snake at boundary.*

**if** (snAKe**.**head**.xcor**() **>=** 360) **or** (snAKe**.**head**.xcor**() **<=** -360) **or** (snAKe**.**head**.ycor**() **>=** 310) **or** (snAKe**.**head**.ycor**() **<=** -310):

        scrBrd**.gameover**()

        game\_on = **False**

    #*Detect collision with tail.*

    #*if head collides with any segment in the tail*

    #*# trigger game.over*

    #*------ can be done with "slicing the list"*

**for** sQr\_segmnt **in** snAKe**.**snake:

**if** sQr\_segmnt **==** snAKe**.**head:

**pass**

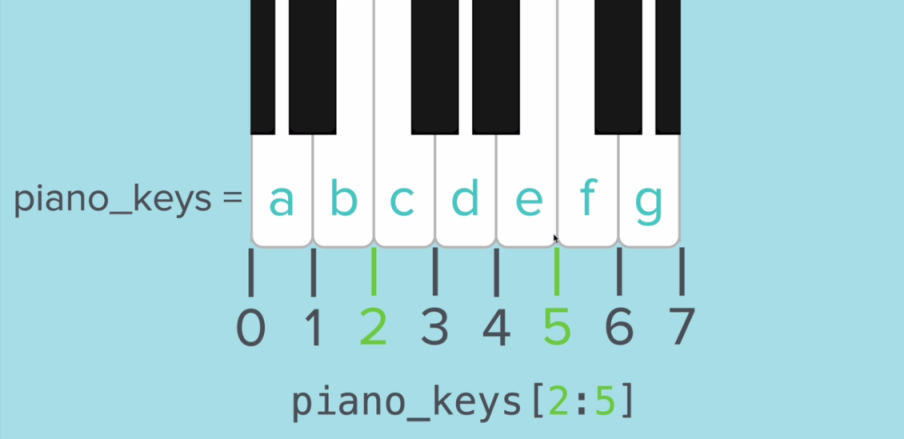
**elif** snAKe**.**head**.distance**(sQr\_segmnt) **<** 10 :

            scrBrd**.gameover**()

            game\_on = **False**

**21.7 SLICING in Python**

Slicing works with ***Tuples*** and ***Lists*** in similar ways.



* Sometimes it looks confusing, because 2 is the position of c but 5 is the position of f . But the slicing is actually works **.**
* The upper element is not-included. (think like this: slicing occurs at "**start**" of the lower-elements position and upper-elements "ending" position, so to include ***e*** we must slice at ***5***)

|  |  |
| --- | --- |
| piano\_keys = ["a", "b", "c", "d", "e", "f", "g"]  **print**(piano\_keys[2:5])  **print**(piano\_keys[2:5:2])  #*python slice\_demo.py*  ***List\_or\_Tuple[start : end : increment]***  ***List\_or\_Tuple[start : ]*** means from ***lower-bound*** to ***last-element*** of the list/tuple. ***increment*** may present.  ***List\_or\_Tuple[ : end]*** means from ***first-element*** to en position to the ***upper-bound***. ***increment*** may present. | piano\_keys = ["a", "b", "c", "d", "e", "f", "g"]  piano\_tuples = ("aa", "bb", "cc", "dd", "ee", "ff", "gg")  **print**(piano\_keys[2:5])  **print**(piano\_tuples[2:5])  #*use the "*increment*"*  **print**(piano\_keys[2:5:2])  **print**(piano\_tuples[2:5:2])  #*for every*seconditem  **print**(piano\_keys[ : : 2])  **print**(piano\_tuples[ : : 2])  #Reverse*the list*  **print**(piano\_keys[ : : -1])  **print**(piano\_tuples[ : : -1])  #*python slice\_demo.py* |

* Now we can apply slicing to our snake-game, and reduce the code.

**for** sQr\_segmnt **in** snAKe**.**snake:

**if** sQr\_segmnt **==** snAKe**.**head:

**pass**

**elif** snAKe**.**head**.distance**(sQr\_segmnt) **<** 10 :

            scrBrd**.gameover**()

            game\_on = **False**

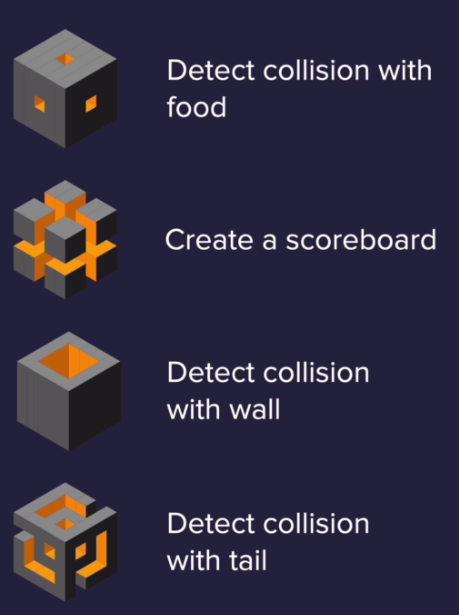
To

**for** sQr\_segmnt **in** snAKe**.**snake[1 : ]:

**if** snAKe**.**head**.distance**(sQr\_segmnt) **<** 10 :

            scrBrd**.gameover**()

            game\_on = **False**



**import** turtle

**import** snake

**import** food

**import** scoreboard

**import** border

#*time module for delay the time*

**import** time

scr = turtle**.Screen**()

#*resizing screen. Setting screen color. Showing title of the sreen*

scr**.setup**(width = 830, height = 700)

scr**.bgcolor**("#a8c64e")

scr**.title**(" ---- The famous NOKIA 1100 snake game ---- ")

#*Turnig off the animation by "tracer" off with "delay" setting*

scr**.tracer**(n = 0)

#*create a snake object*

snAKe = snake**.Snake**()

foOd = food**.Food**()

brDr = border**.Border**()

scrBrd = scoreboard**.Scoreboard**()

#*update the screen after loading the whole snake*

scr**.update**()

scr**.listen**()

scr**.onkey**(snAKe**.**up, "Up")

scr**.onkey**(snAKe**.**down, "Down")

scr**.onkey**(snAKe**.**left, "Left")

scr**.onkey**(snAKe**.**right, "Right")

game\_on = **True**

score = 0

**while** game\_on:

    #*moving the snake segments. first it will move like a Caterpillar. Need to use "update" and disable "tracer". Update screen here when all snake is moved*

    scr**.update**()

    time**.sleep**(0.1)

    snAKe**.move**()

    #*Collition with food*

**if** (snAKe**.**head**.distance**(foOd)) **<=** 10:

        score += 1

**print**("noom noom !!!")

        foOd**.refresh**()

        snAKe**.extend\_body**()

        scrBrd**.icrease\_score**()

    #*snake[0].left(90)*

    #*Wall Collision: Stop the snake at boundary.*

**if** (snAKe**.**head**.xcor**() **>=** 360) **or** (snAKe**.**head**.xcor**() **<=** -360) **or** (snAKe**.**head**.ycor**() **>=** 310) **or** (snAKe**.**head**.ycor**() **<=** -310):

        scrBrd**.gameover**()

        game\_on = **False**

    #*Detect collision with tail.*

    #*if head collides with any segment in the tail*

    #*# trigger game.over*

    #*------ can be done with "slicing the list"*

**for** sQr\_segmnt **in** snAKe**.**snake[1 : ]:

**if** snAKe**.**head**.distance**(sQr\_segmnt) **<** 10 :

            scrBrd**.gameover**()

            game\_on = **False**

#*Screen doesn't disappear autometically*

scr**.exitonclick**()

#*python snake\_main.py*

**import** turtle

STARTING\_POSITION = [(10, 0), (0, 0), (-10, 0)]

MOVE\_DISTANCE = 10

UP = 90

DOWN = 270

LEFT = 180

RIGHT = 0

**class** Snake:

**def** **\_\_init\_\_**(self):

**self.**snake = []

**self.create\_sanke**()

**self.**head = **self.**snake[0] #*This line is called after create\_sanke(), unless error occurs*

    #*create\_sanke(self), add\_segments(self, position) creates the snake body first*

**def** **create\_sanke**(self):

**for** pos **in** STARTING\_POSITION:

**self.add\_segments**(pos)

**def** **add\_segments**(self, position):

            sqrs = turtle**.Turtle**(shape = "square")

            sqrs**.shapesize**(0.4, 0.4, 0)

            sqrs**.color**("#3c412c")

            sqrs**.penup**()

            sqrs**.goto**(position)

                #*By default turtle size is 20 X 20. So ours will be 8 X 8. Distance from each other will be 10.*

**self.**snake**.append**(sqrs)

    #*Extend the body when collide with food*

**def** **extend\_body**(self):

        #*"-1" is used to indicate the "last" element of the list*

**self.add\_segments**(**self.**snake[-1]**.position**())

**def** **move**(self):

        #*Reverse for loop. Move following/backward segments to forward.*

**for** i **in** **range**(len(**self.**snake)-1, 0, -1):

            new\_x = **self.**snake[i-1]**.xcor**()

            new\_y = **self.**snake[i-1]**.ycor**()

**self.**snake[i]**.goto**(new\_x, new\_y)

            #*Shows one by one move "scr.update()". set higher value with higher "delay" to understand. Shows one by one move slowly: time.sleep(1).*

        #*moving the first segment*

**self.**head**.forward**(MOVE\_DISTANCE)

**def** **up**(self):

**if** **self.**head**.heading**() **!=** DOWN:

**self.**head**.setheading**(UP)

**def** **down**(self):

**if** **self.**head**.heading**() **!=** UP:

**self.**head**.setheading**(DOWN)

**def** **left**(self):

**if** **self.**head**.heading**() **!=** RIGHT:

**self.**head**.setheading**(LEFT)

**def** **right**(self):

**if** **self.**head**.heading**() **!=** LEFT:

**self.**head**.setheading**(RIGHT)

#*python snake.py*

**import** turtle

**import** random

**class** Food(turtle**.**Turtle):

**def** **\_\_init\_\_**(self):

**super**()**.\_\_init\_\_**()

**self.shape**("circle")

**self.color**("red")

**self.turtlesize**(0.4, 0.4)

**self.penup**()

**self.speed**("fastest")

**self.refresh**()

**def** **refresh**(self):

        random\_x = random**.randint**(-350, 350)

        random\_y = random**.randint**(-300, 300)

**self.goto**(random\_x, random\_y)

#*python food.py*

**import** turtle

ALIGNMENT = 'center'

FONT = ('Arial', 16, 'normal')

**class** Scoreboard(turtle**.**Turtle):

**def** **\_\_init\_\_**(self):

**super**()**.\_\_init\_\_**()

**self.hideturtle**()

**self.color**("#3c412c")

**self.penup**()

**self.speed**("fastest")

**self.goto**(0, 310)

**self.**score = 0

**self.update\_scoreboard**()

**def** **gameover**(self):

**self.goto**(0 , 0)

**self.write**(f"GAME OVER", move= **False**, align= ALIGNMENT, font= FONT)

**def** **update\_scoreboard**(self):

**self.write**(f"Score : {**self.**score}", move= **False**, align= ALIGNMENT, font= FONT)

**def** **icrease\_score**(self):

**self.**score += 1

**self.clear**()

**self.update\_scoreboard**()

#*python scoreboard.py*

**import** turtle

SCRN = turtle**.Screen**()

**class** Border():

**def** **\_\_init\_\_**(self):

**self.**border\_block = []

        #*Turtle shape can be modified as follows*

        SCRN**.register\_shape**("new\_defined\_1", ((-5, 1), (5, 1), (5, -1), (-5, -1)))

        SCRN**.register\_shape**("new\_defined\_2", ((-1, 1), (1, 1), (1, -1), (-1, -1)))

**self.create\_border**()

**def** **create\_block**(self):

        b\_blok = turtle**.Turtle**()

        b\_blok**.color**("#3c412c")

        b\_blok**.penup**()

        b\_blok**.speed**("fastest")

**return** b\_blok

**def** **create\_border**(self):

**for** i **in** **range**(-354, 354, 2):

**for** up\_or\_down **in** **range**(0, 2):

                brdr\_block = **self.create\_block**()

                brdr\_block**.shape**("new\_defined\_2")

**if** up\_or\_down **==** 0:

                    brdr\_block**.goto**(i, 306)

**else**:

                    brdr\_block**.goto**(i, -306)

**self.**border\_block**.append**(brdr\_block)

**for** j **in** **range**(-306, 306, 2):

**for** up\_or\_down **in** **range**(0, 2):

                brdr\_block = **self.create\_block**()

                brdr\_block**.shape**("new\_defined\_2")

**if** up\_or\_down **==** 0:

                    brdr\_block**.goto**(354, j)

**else**:

                    brdr\_block**.goto**(-354, j)

**self.**border\_block**.append**(brdr\_block)

#*python border.py*