Day - 19

**Instances, State and Higher Order Functions**

Higher order functions (function call wit/without **()**),

function as parameter, Higher Order Functions & Event Listeners,

Object State and Instances, Turtle Coordinate System,

**listen()**,**Turtle.textinput()**

Project: Drawing using Turtle

Project: Turtle Racing

**19.1 Python Higher Order Functions & EVENT LISTENERS**

|  |  |
| --- | --- |
| * Turtle event listener:      1. ***turtle.onkey(fun, key)*** 2. ***turtle.onkeyrelease(fun, key)***   Parameters: ***fun*** a function with no arguments or None  ***key*** a string: key (e.g. ***a***) or key-symbol (e.g. ***space***)  Bind fun to key-release event of key. If fun is None, event bindings are removed.  Note: in order to be able to register key-events, ***TurtleScreen*** must have the focus. (See method ***listen()***.)   * Recall Calculator-Project: * Notice the functions inside the dictionary " operations " there are no pranthesis "**()**" and no parameters are used. ***Only*** ***function*** ***names*** are used.   **operations = {"+" : add, "-" : subtract, "\*" : multiply, "/" : devide}** | num1 = **int**(input("Enter the first number : "))  num2 = **int**(input("Enter the second number : "))  #*Add*  **def** **add**(n1, n2):  **return** n1 + n2  #*subtract*  **def** **subtract**(n1, n2):  **return** n1 - n2  #*Multiply*  **def** **multiply**(n1, n2):  **return** n1 \* n2  #*Devide*  **def** **devide**(n1, n2):  **if**(n2**>**0):  **return** n1 / n2  **else**:  **return** "undefined"  **operations = {"+" : add, "-" : subtract, "\*" : multiply, "/" : devide}**  **for** op **in** operations:  **print**(op)  oPr = **input**("\n\t Pick an operation from above : ")  result = operations[oPr](num1, num2)  **print**(f"\n\t{num1} {oPr} {num2} = {result}\n")  **# or we could use following function** |
| **19.2 Function as Inputs**   * Notice no ***()*** pranthesis is used in the parameters of ***onkey()***.   scrn**.onkey**(fun = move\_fwrd, key = ".")   * With **()** function is executed immediately. * To use a function inside of other function "and we want it to execute later" use it without **()**.   **def** **function\_a**(something):      #*Do this with something*      #*#Then do this*      #*#Finally do this*  **def** **function\_b**():      #*Do this*  #*Functions as Inputs*  #*No arguments are used with function\_b*  **function\_a**(function\_b) | **import** turtle  tom = turtle**.Turtle**()  scrn = turtle**.Screen**()  scrn**.colormode**(255)  **def** **move\_fwrd**():      tom**.forward**(10)  #*Listen for an event*  scrn**.listen**()  #*Bind an event to the event-listner*  scrn**.onkey**(move\_fwrd, "f")  scrn**.exitonclick**() |

* Higher order-function: A function is called Higher Order Function (HOF) if it contains other ***functions*** as a ***parameter*** or ***returns*** a ***function*** as an ***output***
* Exercise 19.1: Challenge Make an Etch-A-Sketch App.

|  |  |
| --- | --- |
|  |  |
| Practiced Version | Instructor Version |
| **import** turtle  tom = turtle**.Turtle**()  scrn = turtle**.Screen**()  scrn**.colormode**(255)  **def** **move\_fwrd**():      tom**.forward**(10)  **def** **move\_bkrd**():      tom**.back**(10)  **def** **move\_counter\_clockwise**():      tom**.circle**(100.0, -10)  **def** **move\_clockwise**():      tom**.circle**(100.0, 10)  **def** **clear\_drawing**():      tom**.setposition**(0.0, 0.0)      tom**.clear**()  #*turtle.circle(120, 180)  # draw a semicircle*  #*Listen for an event*  scrn**.listen**()  #*Bind an event to the event-listner*  scrn**.onkey**(fun = move\_fwrd, key = "w")  scrn**.onkey**(fun = move\_bkrd, key = "s")  scrn**.onkey**(fun = move\_counter\_clockwise, key = "a")  scrn**.onkey**(fun = move\_clockwise, key = "d")  scrn**.onkey**(fun = clear\_drawing, key = "c")    scrn**.exitonclick**()  #*python evevnt\_lstnr\_turtle\_scketh.py* | **import** turtle  tom = turtle**.Turtle**()  scrn = turtle**.Screen**()  scrn**.colormode**(255)  **def** **move\_fwrd**():      tom**.forward**(10)  **def** **move\_bkrd**():      tom**.back**(10)  **def** **move\_counter\_clockwise**():      tom**.right**(10)      #*Also:  new\_heading = tom.heading() - 10*      #*tom.setheading(new\_heading)*  **def** **move\_clockwise**():      #*Also: tom.left(10)*      new\_heading = tom**.heading**() + 10      tom**.setheading**(new\_heading)  **def** **clear\_drawing**():      tom**.penup**()      tom**.home**() #*tom.setposition(0.0, 0.0)*      tom**.clear**()      tom**.pendown**()  #*turtle.circle(120, 180)  # draw a semicircle*  #*Listen for an event*  scrn**.listen**()  #*Bind an event to the event-listner*  scrn**.onkey**(fun = move\_fwrd, key = "w")  scrn**.onkey**(fun = move\_bkrd, key = "s")  scrn**.onkey**(fun = move\_counter\_clockwise, key = "a")  scrn**.onkey**(fun = move\_clockwise, key = "d")  scrn**.onkey**(fun = clear\_drawing, key = "c")  scrn**.exitonclick**()  #*python evevnt\_lstnr\_turtle\_scketh\_soln.py* |

**19.3 Instances & State**

* Instances : Different *objects* of same *class* are separate things in Python (what about assignment & reference ?). These objects are called ***instances*** of that class.
* State: Different objects of as *same* *class* can do separate things at same time these are called ***states*** of those objects.

**19.4 Turtle coordinate-system**

Origin is located at (bredth/2, height/2) considered as (0, 0).

* Exercise 19.1: Turtle Racing part-1 : "Assemble Turtles"

Practiced Version

**import** turtle

scrn = turtle**.Screen**()

colors = ["red", "orange", "yellow", "green", "blue", "purple", "black"]

#*use setup() instead of screensize()*

scrn**.setup**(1200, 650)

user\_guess = scrn**.textinput**("Guess the Turtle", "Guess which Turtle gonna win: ")**.lower**()

#*can use tom.setposition(a, b)*

all\_turtls = []

vert\_pos = -225

**for** colr **in** colors:

    tom = turtle**.Turtle**()

    tom**.penup**()

    tom**.shape**("turtle")

    tom**.color**(colr)

    tom**.goto**(-575, vert\_pos)

    vert\_pos += 75

    all\_turtls**.append**(tom)

**print**(user\_guess)

scrn**.exitonclick**()

#*python turtle\_race.py*

Instructor Version

y\_pos = [-225, -150, -75, 0, 75, 150, 225]

all\_turtle = []

**for** turtle\_index **in** **range**(0, 7):

    #*Notice we are actually declaring "Turrle" object with "shape"*

    new\_turtle = turtle**.Turtle**(shape= "turtle")

    new\_turtle**.color**(colors[turtle\_index])

    new\_turtle**.penup**()

    new\_turtle**.goto**(x = -575, y = y\_pos[turtle\_index])

    all\_turtle**.append**(new\_turtle)

**print**(user\_guess)

**if** user\_guess:

    game\_on = **True**

**while** game\_on:

**for** trtl **in** all\_turtle:

        random\_number = random**.randint**(0, 10)

        trtl**.forward**(random\_number)

**if** trtl**.position**()[0] **>=** 580:

            game\_on = **False**

**print**(f"The \' {trtl**.color**()} \'Turtle wins")

#*notice how y-position is retrived from the returned "tuple" from "fromposition()"*

#*print(all\_turtle[6].position()[1])*

* Retrieve the function returned tuple:

#*notice how y-position is retrived from the returned "tuple" from "fromposition()"*

**print**(all\_turtle[6]**.position**()[1])

* Retrieve the turtle-color: trtl**.color**() returns ("color", "color"). One is ***pencolor*** and other is ***fillcolor*** as tuple. Can be accessed bu using **pencolor**() or **fillcolor**()

***turtle.color(\*args)***

Return or set *pencolor* and *fillcolor*. So we need to use trtl**.pencolor**().

Instructors solution

**import** turtle

**import** random

scrn = turtle**.Screen**()

colors = ["red", "orange", "yellow", "green", "blue", "purple", "black"]

#*use setup() instead of screensize()*

scrn**.setup**(1200, 650)

user\_guess = scrn**.textinput**("Guess the Turtle", "Guess which Turtle gonna win: ")**.lower**()

y\_pos = [-225, -150, -75, 0, 75, 150, 225]

all\_turtle = []

**for** turtle\_index **in** **range**(0, 7):

    #*Notice we are actually declaring "Turrle" object with "shape"*

    new\_turtle = turtle**.Turtle**(shape= "turtle")

    new\_turtle**.color**(colors[turtle\_index])

    new\_turtle**.penup**()

    new\_turtle**.goto**(x = -575, y = y\_pos[turtle\_index])

    all\_turtle**.append**(new\_turtle)

**print**(f"You choose : {user\_guess}")

**if** user\_guess:

    game\_on = **True**

#*match fixing*

all\_turtle[3]**.forward**(100)

**while** game\_on:

**for** trtl **in** all\_turtle:

        random\_number = random**.randint**(0, 10)

        trtl**.forward**(random\_number)

**if** trtl**.position**()[0] **>=** 590:

            game\_on = **False**

            winner\_color = trtl**.pencolor**()

**print**(f"The \' {trtl**.pencolor**()} \'Turtle wins")

**if** winner\_color **==** user\_guess:

**print**("you win the bet")

**else**:

**print**("You lost the bet :( ")

#*notice how y-position is retrived from the returned "tuple" from "fromposition()"*

#*print(all\_turtle[6].position()[1])*

scrn**.exitonclick**()

#*python turtle\_race.py*

* Expand on the Solutions: Given Solutions are simple enough, you can expand/modify those solutions.