GUpPy: Graphics Unplugged (Python) Syntax

PHYSICAL OBJECTS: Visualization Grid, 20 Tiles (10 pink, 10 yellow), 2 Game Pieces, 2 Die, 4 Dry-Erase Cards for storing variables (x,y,i,d), Dry-erase marker.

KEYWORDS that REPRESENT PHYSICAL OBJECTS and ACTIONS

PINK: represents a physical tile colored pink (any of the pink tiles)
YELLOW: represents a physical tile colored yellow (any of the yellow tiles)
RANDOM: represents a physical tile of random color (that the user randomly selects)
PIN1: represents a game piece (a specific colored game piece) that pins a location
PIN2: represents the other game piece (i.e. a different color) that pins a location
UP: the direction to move the game piece (increasing the row)
DOWN: the direction to move the game piece (decreasing the row)
RIGHT: the direction to move the game piece (increasing the column)
LEFT: the direction to move the game piece (decreasing the column)
if: part of conditional statement. Starts a block of code to be executed when the condition is true
else: part of an if-statement that starts the block of code executed when the condition is false
( ): used to signify a function is being called (i.e. an action is being performed)
( ): also used to show precedence among operators
Arithmetic operators: `+ - * /`
Relational operators: `< > <= >= ==`
Assignment operator: =

Key words and symbols	Action / Description	`Code Example(s)`

Tile(*color*)	Grab a new tile based on the color. If RANDOM, grab a tile from the bag without looking.	`Tile(RANDOM)`
		`tile = Tile(PINK)`

Tile(*color*).place(*col*,*row*)	Place the specified tile at grid location (col,row).	`Tile(PINK).place(2,8)`
pin.place(*col*,*row*)	Place specified pin at grid location (col,row).	`PIN1.place(2,12)`
tile.place(*pin*)	Place specified tile at location of specified pin.	`Tile(RANDOM).place(PIN1)`

pin.shift(*dir*,*squares*)	Move the specified game piece in the specified direction and number of grid squares.	`PIN1.shift(UP,3)`
		`PIN2.shift(LEFT,1)`

variable = value
 	
(Assignment Statement)
	Assign value to the variable. *Use your dry-erase cards to record the value of variables, such as variables x,y,d,i,col,row*	`tile = Tile(PINK)`
		`x = 5`
		`y = roll()`

tile.getColor()	**EXPRESSION** that evaluates to the color of the specified tile.	`tile = Tile(RANDOM)`
		`color = tile.getColor()`

roll()	Physically roll die to get a random number between 1 and 6**, inclusive**	`x = roll()`
		`Y = roll() + roll()`

value REL_OP value	**BOOLEAN EXPRESSION** that evaluates to True or False. It compares the values using the relational operator.	`roll() < 4`
		`x >= 3`
		`6 == i`

if (Boolean expression):
  code block
	IF-STATEMENT that starts a code block that is executed whenever the Boolean expression is True. It may or may not include an "else" block. Note the indentation – it is important!	`if x >= 12:`
		    `x = x - 1`

if (Boolean expression):
  code block (True)
else:
  code block (False)
	IF-ELSE STATEMENT that provides 2 possible blocks of code to execute, depending on the value of the Boolean expression. Note the indentation – it is important!	`if x >= 12:`
		`   x = -1`
		`else:`
		`   x = +1`

GUpPy: Graphics Unplugged (Python) Additional Syntax

while:	part of a conditional statement that tests a Boolean expression and starts a block of code that is repeated for as long as the condition is true
true:	Boolean value
false:	Boolean value
not:	operator applied to either true or false, which evaluates to the opposite value
in:	operator that tests if a value is in a list
and:	joins 2 Boolean expressions, returns true if both expressions are true. Returns false if either expression is false (or both are false).
or:	joins 2 boolean expressions, return true if either expression is true
print:	“displays” values (print on a piece a piece of paper)
[ ]:	holds a list of elements, such as numbers, variables, tiles, etc.

Arithmetic operators: + - * / %

Relational operators: < > <= >= == in

Boolean operators: and or not

Assignment operator*:* =

FUNCTIONS / OPERATIONS with EXTENDED KEYWORDS	**Code Example(s)**

while ***(Boolean expression)***: begins a code block that is repeatedly executed until the condition is false	`while x <= 12:`
	`   x = x + 1`

(Boolean expression) and (Boolean expression): evaluates to true if both expressions are true, otherwise it evaluates to false	`if x>=0 and x<12:`
	`   tile.place(x,5)`

(Boolean expression) or (Boolean expression): evaluates to true if either condition is true.	`while x<3 or x>6:`
	`   x = roll()`

value in list: evaluates to true if the value is in the list	`if roll() in [1,3,5]:`
	`   x = -1`

not (Boolean expression): evaluates to the opposite of the value of the condition. In other words *not true* is false; *not false* is true.	`if not y<0:`
	`  tile.place(6,y)`

print(value): write the value on a piece of paper	`print(tile.color())`
	`print(x)`