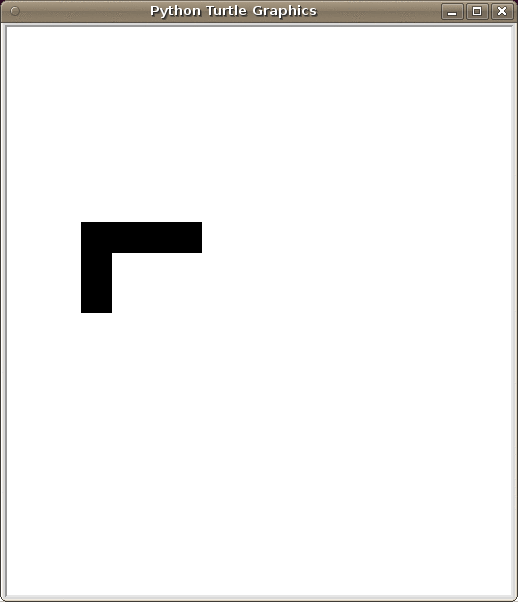


**π3**

**Python, the Python python**

Today, you will make “Python”, the MEET version of the popular Snake arcade game.

The completed version looks like this:

Let’s get started!

# Part 0: What’s a turtle? What’s Turtle?

*Turtle is a graphics module for Python. It lets you draw things on your screen!*

Instructions, Hints:

1. **Create a folder called mini\_project**
2. **Inside the folder,put the code below in a file called part0.py**
3. Complete the three functions below
   1. DO NOT use .forward(). Instead, use .setpos()
   2. Use a for-loop in draw\_square()
   3. HINT: For draw\_square() make a list of the points we need
4. Check your code by running it in IDLE shell. Look below for what you should type in
5. Try running your code without .pendown() or .hideturtle(). What happened?

|  |
| --- |
| import turtle  turtle.pendown()  turtle.hideturtle()  def vert\_line():  '''  Draws a vertical line on the turtle canvas.  '''  #insert code here    def horz\_line():  '''  Draws a horizontal line on the turtle canvas.  '''  #insert code here  def draw\_square():  '''  Draws a square on the turtle canvas.  '''  #insert code here |
| >> #test your code  >> vert\_line()  >> horz\_line()  >> draw\_square() |

# Part 1: Tick, tock, turtle timers!

*Timing is everything.*

Instructions, Hints:

1. Create a file called part1.py with the code below.
   1. Read .ontimer() and .mainloop() on your reference sheet.
   2. Try running the file. Watch the IDLE shell window!
2. Modify the code so we say\_hi after 1 second.
3. Modify your say\_hi function so we say\_bye 3 seconds after we call say\_hi.
4. Modify the code so we say\_hi every second.
5. Complete .countdown(). Your output should look like ours below.

|  |
| --- |
| import turtle  turtle.penup() #no more pen trails  turtle.ht()  COUNTDOWN = 10  def say\_hi():  print("Hi MEET student!")  def say\_bye():  print("Bye MEET student!")  def countdown():  '''  Prints a countdown from 10.  '''  global COUNTDOWN  turtle.ontimer(say\_hi,3000) #don’t remove this line  turtle.mainloop() |
| >> countdown()  >> 5  >> 4  >> 3  >> 2  >> 1  >> 0 |

# Part 2: Listeners and user inputs.

*How can you play a game without a mouse or keyboard?*

Instructions, Hints:

1. Create an empty file called part2.py
   1. Read about .mainloop(), .listen() and .onkey() on your reference sheet.
   2. **Set up part2.py for Turtle. Remember .penup(), .hideturtle(), .import()**
2. Define 4 strings.
   1. UP = 'You pressed Up!'
   2. DOWN = 'You pressed Down!'
   3. Do the same for LEFT and RIGHT.
3. Define a constant STEPSIZE = 500.
4. Create a variable direction. Initialize it as LEFT.
5. Create 4 functions, up(), down(), left(), right(). Each function should:
   1. Set direction to the string you made before.
   2. What rhymes with mobile, noble, and lobal?
6. Map the 4 functions to your arrow keys, using .onkey()
7. Create a function repeat(). It should:
   1. Print the variable direction
   2. Repeat by STEPSIZE. HINT:.ontimer()
8. **End your file with .listen()**
9. Run part2.py. Call repeat().
   1. What does .onkey() do?
   2. What happens when you remove .listen()?
   3. What happens when you remove .mainloop()?

Your output should look like something like this:

|  |
| --- |
| >>> repeat()  You pressed Left!  >>> You pressed Left!  You pressed Left!  You pressed Left!  You pressed Right!  You pressed Down!  You pressed Up! |

# Part 3: Drawing/erasing on Turtle canvas.

*Let’s put more things on our canvas!*

Instructions, Hints:

1. Create a file called part3.py with the code below.
2. Read about .shape(), .stamp(), and .clearstamp() on your reference sheet.
3. Change your shape to a square.
4. HINT: Rhymes with mobile and noble.
5. You can store stamp ID’s in a variable.
   1. Use STAMP\_ID for the first 2 functions. Use STAMP\_IDs for the next 2.

|  |
| --- |
| import turtle  turtle.pu()  turtle.ht()  ## Change turtle to a square here.  STAMP\_ID = 0  STAMP\_IDs = []  def stamp\_block():  '''  Places a block in the center of the canvas.  '''  ## HINT  STAMP\_ID = turtle.stamp()  def erase\_block():  '''  Erases the block placed by stamp\_block().  '''  ## HINT  def big\_rectangle(x,y):  '''  Creates an x by y rectangle of stamps.  '''  def erase\_rectangle():  '''  Erases big\_rectangle.  ''' |

# 

# 

# Part 4: Animation!

*What fun is a game where everything stays still?*

Instructions, Hints:

1. Create a file called part4.py with the code below.
   1. Notice that your direction constants (UP,DOWN,...) have changed. Does this matter?
2. Run flashing\_block() in IDLE shell.
   1. What’s happening? Why?
   2. What happens if you remove global?
3. Complete left\_block().
   1. You only need to change ONE line of code from flashing\_block().
   2. [No code] How would you write right\_block()? down\_block()? up\_block()?
4. Complete controlled\_block(). Test it out in shell like below.

|  |
| --- |
| import turtle  turtle.pu()  turtle.ht()  turtle.shape("square")  BLOCK\_POS = (0,0)  STAMP\_ID = 0  UP,DOWN,LEFT,RIGHT = 0,1,2,3  def flashing\_block():  global STAMP\_ID  global BLOCK\_POS    turtle.clearstamp(STAMP\_ID)  nx,ny = BLOCK\_POS  turtle.setpos(nx,ny)  STAMP\_ID = turtle.stamp()    turtle.ontimer(flashing\_block,100)  def left\_block():  '''  Makes a block move to the left,  until it disappears off the screen  '''  def controlled\_block(DIRECTION):  '''  Moves a block directly left, right, up,  or down from where it was before,  depending on DIRECTION  ''' |
| >> #test your code like this. watch your turtle canvas.  >> controlled\_block(UP)  >> controlled\_block(DOWN) |

# Part 4: Animation! (continued)

*What fun is a game where everything stays still?*

Instructions, Hints:

1. Create a variable DIRECTION
2. Set up .onkey() for the arrow keys.
   1. Each function should set DIRECTION to the appropriate constant (UP,DOWN,etc…)
   2. Here’s a short reminder:

|  |
| --- |
| def up():  #mobile? noble? lobal?  turtle.onkey(up,"Up")  turtle.listen() |

1. Modify controlled\_block() to respond to arrow keys. **First, ask yourself:**
   1. How does the function know which direction to set?
   2. Do I still need the DIRECTION argument?

Now we’re getting somewhere! When it’s all done, it should look like:

# Part 5: Powerful Python, Slippery Snake

*Sssssssssssssssss...*

Instructions, Hints:

1. Create a file called snake.py with the code below.
   1. Read about .clone() and .clearstamps() on your reference sheet.
2. How does a Snake move?
   1. Do we have to erase and draw every block as it moves?
   2. **HINT: First and last blocks.**
3. Now we’ll create a multi-block snake. In the for-loop:
   1. Stamp using snake.stamp(), NOT turtle.stamp()
   2. Append block\_pos with the current position of snake
   3. Move snake to the right.
4. Set up your listeners for arrow-keys.
5. Copy your code from controlled\_block() to move\_snake().
6. Modify the function to work with your new Snake.
   1. The snake starts out moving RIGHT.
      1. Which block is the head?
      2. Which block is the tail?
      3. Where are the head and tail positions in block\_pos?
   2. Don’t store stamp ID’s. Instead use .clearstamps()
      1. Which block did your for-loop stamp first? Last?
      2. Which one should you remove?

|  |
| --- |
| import turtle  turtle.pu()  turtle.ht()  turtle.shape("square")  block\_pos = []  UP,DOWN,LEFT,RIGHT = 0,1,2,3  SNAKE\_LENGTH = 5  DIRECTION = RIGHT  snake = turtle.clone()  for i in range(SNAKE\_LENGTH):  ##code to initialize snake here  def move\_snake():  '''  Moves a multiple-block Snake.  ''' |

# Part 6: BUMP! CRASH! Collision detection.

*Don’t be running into things now...*

Instructions, Hints:

1. In snake.py, create a function called collide\_detect(x,y). It should…
   1. Take the snake’s head position as x,y
   2. Check whether the head is touching a wall
   3. Check whether the snake is touching itself (HINT: Use block\_pos).
   4. Return True if the head is touching, False if not.
2. See the graphic below. Where should collide\_detect go?
3. Use collide\_detect() in move\_snake().
   1. If collide\_detect() returns False, the snake should move normally.
   2. If collide\_detect() returns True, print “Game Over!”

