WELCOME
CIS 210 COMPUTER SCIENCE I

PROGRAMMING GUIDE – FALL 2020 WEEK 1
CIS 210 Fall 2020
Guide for CIS 210 Projects - Week 1

Project 1a “UO guide”

Turtle graphics
Python Turtle graphics, inherited from the computer programming language Logo, gives us a graphics (and simple user interface) programming tool. From plotting data to creating unique pieces of art, we will make use of the graphical output in several problems this term.

The Python “Turtle” is essentially a robot that is controlled with Python code. The Turtle/robot comes equipped with multiple attributes, such as position, heading, color, and size.

We will be making use of the the anonymous turtle for this project. This means you don’t need to set up the Turtle ( t = turtle.Turtle() ) like in the book. Commands can be called directly, after using ‘from turtle import *’ (leave off the quotes).

Turtle graphics commands you will need for Project 1a:
The following Turtle commands are all you need for P1a. If a command is not listed in the project 1a section, do not use it for project 1a. Make sure to try out the commands in the shell to aid in understanding them.

fd(distance) / forward(distance): Move the Turtle forward by the specified distance, in the direction the Turtle is headed.

```
>>> from turtle import *
>>> fd(60)
```

Try it yourself in the shell:

```python
>>> from turtle import *
>>> fd(60)
```

Note how the command used is ‘fd(60)’ and not ‘turtle.fd(60)’ We are able to do this because we imported everything from the turtle module (from turtle import *). You will never need to preface any of your commands with turtle. (i.e. turtle.command()) for this project.
**bk(distance) / back(distance) / backward(distance):** Move the Turtle backward by distance, opposite to the direction the Turtle is headed. Does not change the Turtle’s heading.

Try it yourself in the shell:
```python
>>> from turtle import *
>>> bk(60)
```

**lt(angle) / left(angle):** Turn the Turtle left by angle units, relative to the Turtle’s current heading.
(By default, the unit is degrees)

Try it yourself in the shell:
```python
>>> from turtle import *
>>> lt(90)
```

**rt(angle) / right(angle):** Turn the Turtle right by angle units, relative to the Turtle’s current heading.
(By default, the unit is degrees)

Try it yourself in the shell:
```python
>>> from turtle import *
>>> rt(90)
```
Now combine them!
Look at the code below, try drawing what you think will happen, then execute the code to see:

```python
>>> from turtle import *
>>> fd(100)
>>> lt(90)
>>> bk(100)
>>> lt(90)
>>> fd(100)
>>> rt(270)
>>> bk(100)
```

What happens?

**stamp()**: Stamp a copy of the Turtle shape onto the canvas at the current Turtle position.

```
>>> from turtle import *
>>> stamp()
>>> fd(60)
```

**dot()**: Puts a dot onto the canvas at the current Turtle position.

Note the similarity to stamp, the only difference is the shape being left.

```
>>> from turtle import *
>>> dot()
>>> fd(60)
```
More Turtle commands:
These Turtle commands are included in the Project 1a starter code. You can simply execute the starter code to use these commands, but you may find it interesting to explore them, too.

reset(): Deletes the Turtle’s drawings from the screen, re-centers the Turtle and sets variables (pen size, speed, etc...) to their default values.

```
>>> from turtle import *
>>> fd(20)
>>> rt(90)
>>> fd(20)
>>> reset()
```

clear(): Deletes the Turtle’s drawings from the screen. Does not move or rotate the Turtle.

```
>>> from turtle import *
>>> fd(20)
>>> rt(90)
>>> fd(20)
>>> clear()
```

title(titlestring): sets title of Turtle window to titlestring.

```
# Python Turtle Graphics
Top of turtle window before a title is specified

# Welcome to Computer Science at the UO!
Specify a title

# Welcome to Computer Science at the UO!
Top of turtle window after a title is specified
```

Try it yourself in the shell:
```python
>>> from turtle import *
>>> title('Welcome to Computer Science at the UO!')
```
**speed(newspeed):** sets the Turtle’s speed to an integer value in the range 0 – 10. Strings can also be used to set the speed (see below).

- “fastest”: 0
- “fast”: 10
- “normal”: 6
- “slow”: 3
- “slowest”: 1

speed(“fast”) is the same as speed(10), speed(“slow”) is the same as speed(3), etc...

**bgpic():** Sets background image.
Note: image file must be in same folder as .py file

**pencolor(colorstring):** Sets the pen color. Note: once the pen color changes this only affects lines drawn after the color change occurred. Previously drawn lines will keep the same color that they were originally drawn in.

Try it yourself in the shell:
>>> from turtle import *
>>> fd(30)
>>> pencolor(“red”)
>>> fd(30)
`screensize(canvaswidth, canvasheight)`: If no arguments are given, returns the width and height of the canvas, else resizes the canvas the Turtle is drawing on.

Try it yourself in the shell:

```python
>>> from turtle import *
>>> screensize(1000, 1000)
```
**Python**

These Python keywords/and syntax are included as part of the starter code, and will continue to be core elements of our Python programming.

**from Turtle import ***: ‘from Turtle’ means we are importing content from the Turtle library. ‘*’ means we want to import everything in the library, so we can directly use it without having to add “Turtle.” to the start of what we are using. i.e. `fd(100)` will make the Turtle move, as opposed to having to type `Turtle.fd(100)`

from and import are Python keywords

**def**: `def` is a Python keyword that marks the start of a function header. The pattern for creating a function being: `def`, function name, parameter list inside parentheses (though if no parameters are needed, the parenthesis are just left empty)

**Comments**: a comment is code that programmers can read when looking at a file, but is not executed by the Python interpreter.

`#` a hash mark like this indicates a single line comment

```
```

Multi-line comments are made by typing between two sets of triple quotes

```
```

``````

Double quotes work as well
Just make sure there are 3 before, and 3 after.
```

**pass**: `pass` is a null operation, when it is executed nothing happens.
It is generally used as a placeholder for where a statement is required by syntax, but no code is needed to be executed, or a programmer later wants to come back and write code at that place.

Is a Python keyword.

**return**: indicates the end of the function. Is a Python keyword
(1) The file header comments provide information about this Python file (program):

```python
... Title: a one-line description/title for the program
Author: Your name
Credit: reference any other sources (materials, people, etc.) for this work
...
```

(2) The import statement provides access to turtle module functions when the function is executing. Import is a Python keyword. By convention, import statements appear at the top of the program file, after the file header and before the rest of the code.

```python
from turtle import *
```

(3) Note the structure of the Python `uo_guide` function:

```python
def uo_guide_start():
    ...
    Welcome to the UO! Welcome to Computer Science!
    Guide students from the EMU Lawn to Dechutes Hall, home of the
    Computer Science Department, and then to Price Science Commons
    (Science Library), home of B004/A computer lab and study space.
    >>> uo_guide_start()
    ...
    # setting the scene (supply this code)
    reset()
    clear()
    title('Welcome to Computer Science at the UO!')
    color('purple')
    pensize(3)
    speed('slowest')
    bgpic('uo_campus_map.png')
    screensize(1195, 488)
    stamp() #mark start of route on EMU East lawn
    # replace pass with your code
    # guide to Dechutes
    pass
    # guide to Price Science Commons
    pass
    return
```

- **Function header:**
  - `def function name`: defines a function with the provided name.
  - Parameter list inside parentheses (empty here, `uo_guide` has no parameters)

- **Function docstring:**
  - Inside triple quotes – a brief description of the function followed by an example call to the function

- **Function code:**
  - Some is provided for you; you will write the rest

- **Return statement:**
  - Indicates the end of the function. `return` is a Python keyword.
Project 1b “Art Show”
You’ve seen that Turtle can be used to create maps to aid others in the finding of specific locations. What else can Turtle be used for, though? What about in the creation of art?

Turtle graphics
Turtle graphics commands you will need for Project 1b:
The following Turtle commands, as well as the commands we learned in P1a, are all you need for P1b. if a command is not listed in the P1a or P1b section, do not use it for this project.
Make sure to use the shell to aid in understanding the commands.
We will once again use the anonymous turtle, as well as directly using commands (i.e. just call command(), not turtle.command() )

**fillcolor('colorname')**: Sets the fill color for when the Turtle draws a shape. This may seem like the pencolor command, but pencolor changes the color of the lines drawn, where fillcolor decides the color of the negative space between the drawn lines.

Let’s try it!

Consider the following code, try figuring out what you think it will create, and then check it in the shell!

```python
>>> from turtle import *
>>> fillcolor('blue')
>>> begin_fill()
>>> fd(100)
>>> lt(120)
>>> fd(100)
>>> lt(120)
>>> fd(100)
>>> end_fill()
```

Were you right?
More Turtle commands:
These Turtle commands are included in the Project 1b starter code. You can simply execute the starter code to use these commands, but you may find it interesting to explore them, too.

**pu() / penup():** Picks the Turtle pen up, so that if the Turtle moves, it will not draw.

```
>>> from turtle import *
>>> fd(30)
>>> pu()
>>> fd(30)
```

**pd() / pendown():** Puts the Turtle pen down, so if the Turtle moves, lines will be drawn.

```
>>> from turtle import *
>>> fd(20)
>>> pu()
>>> fd(20)
>>> pd()
>>> fd(20)
```
**hideTurtle():** Hides the Turtle arrow, so that only the marks the Turtle draws are seen.

![Hide Turtle Arrow]

<table>
<thead>
<tr>
<th>Position of turtle:</th>
<th>setpos(20, -20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td></td>
</tr>
</tbody>
</table>

Try it yourself in the shell:
```python
>>> from turtle import *
>>> hideturtle()
>>> fd(60)
```

**setpos(x, y):** sets the new position for the Turtle. Lines get drawn between the old and new position, if the Turtle pen is down. Note that the new Turtle position is independent of its prior position, as opposed to fd/bk which move relative to the Turtle’s current position.

![Set Position]

<table>
<thead>
<tr>
<th>Position of turtle:</th>
<th>setpos(20, -20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(20, -20)</td>
<td></td>
</tr>
</tbody>
</table>

Try it yourself in the shell:
```python
>>> from turtle import *
>>> setpos(20, -20)
```

**pensize(size):** sets the size of the line the Turtle draws.

![Pensize]

| fd(30) | pensize(3) | fd(30) |

Try it yourself in the shell:
```python
>>> from turtle import *
>>> fd(30)
>>> pensize(3)
>>> fd(30)
```
**Python**

*for loop*: Often when writing code, you will find that you end up writing the same line multiple times over, this is when a for loop comes in handy. A For loop can be used to repeat the same line of code multiple times over.

Here is an example of the type of for loop you will need to complete project 1b:

```python
for i in range(4):
    print(i)
```

Keyword ‘for’ that tells python we want to make a for loop

Set a variable name in this case ‘i’, can be any valid variable name

Denote how much our loop will run, in this case 4 times. The value of `i` will be 0, 1, 2, or 3 depending on loop iteration

Code indented after the colon will be executed each time the loop runs

Try typing this for loop into your own shell and playing around with it.
What happens when you change the number within range?
What happens when you change ‘i’ to different variable names?
Maybe try adding more prints, or other lines of code to run within the loop.

**Assignment**: we saw the use of the variable ‘i’ in the above for loop, but we can also assign values to variables outside of for loops using the ‘=’ operator. For example, `x = 45` or `course = ‘CIS210’`

Try these examples in the shell:
```bash
>>> x = 45
>>> x
```

What is printed in the shell?

```bash
>>> course = ‘CIS210’
>>> course
```

What is printed in the shell?

```bash
>>> pi = 3.14
>>> pi
```

What is printed in the shell?
Writing Functions: for this project you will write your own functions. Start by looking at the functions already provided to you with the project starter code from P1a as well as P1b, and then try writing your own with a similar structure, but with different contents. Functions generally have 4 main sections:

1 the function header
def name(parameters if any, will be empty for this assignment):
2 the docstring
   
   Information about the function/what the function does
   Examples of use
   
3 the code within the function
   pass
4 the return marking the end of the function
   return

Project 1c “Art Show Better”
We’ve made some great artwork with “Art Show”, but the program really can only make one picture, and that picture is hard to tweak. Function Parameters can be used to make our picture easier to modify.

Turtle graphics
Turtle graphics commands you will need for Project 1c:
There are no new commands, we will be using the same commands we learned in P1a, and P1b. If a command is not listed in the P1a or P1b section, do not use it for this project.
We will once again use the anonymous turtle, as well as directly using commands (i.e. just call command(), not turtle.command() )

Python
Function parameters: In the past we have left the parentheses of our functions empty, we will now be putting variables into our parentheses. These are called function parameters. A function parameter allows you to pass a variable into a function when the function is called, this is called passing an argument to the function. As an example, let’s say we want a function that has the Turtle draw a line of a provided distance, and then return to where it started. How do you get the number for the distance into that function? You can use a parameter:

def draw_line_and_return(distance):
   
   Draws a line the length of “distance” then returns the Turtle to its starting point
   >>> draw_line_and_return(40)
   
   fd(distance)
   pu()
   bk(distance)
   pd()
   return
Here our parameter is ‘distance’ note how instead of having a hardcoded number, ‘distance’ is used for fd and bk.

But what happens if someone calls the function without an argument (e.g. draw_line_and_return())? Currently the function will just throw an error, but functions can also have Default Parameters.

A Default Parameter is a parameter that is set by the programmer, which a function can use if the user does not provide an argument.

For the previous example function, let’s change it so if the user calls the function without an argument, instead of having an error, 20 will be used:

def draw_line_and_return(distance=20):
    '''
    Draws a line the length of “distance” then returns the Turtle to its starting point
    >>> draw_line_and_return(40)
    >>> draw_line_and_return()
    '''
    fd(distance)
    pu()
    bk(distance)
    pd()
    return

Note in the function header how instead of (distance) it is now (distance=20), the ‘=20’ is the key to setting the default value. What this says is that if a user doesn’t provide a value for ‘distance’, then ‘distance’ is, by default, set to 20.

Try it yourself, what happens if you change 20 to 60?
What about other numbers?
Can you have multiple default parameters?
Turtle Commands

- `fd / forward` .... project 1a, page 1
- `bk / back` ........ project 1a, page 2
- `lt / left` ........... project 1a, page 2
- `rt / right` .......... project 1a, page 2
- `stamp` ............. project 1a, page 3
- `dot` ............... project 1a, page 3
- `reset` ............. project 1a, page 4
- `clear` ............. project 1a, page 4
- `title` ............. project 1a, page 4
- `speed` ............. project 1a, page 5
- `bgpic` ............. project 1a, page 5
- `pencolor` .......... project 1a, page 5
- `screensize` ....... project 1a, page 6
- `fillcolor` .......... project 1b, page 9
- `begin_fill` ....... project 1b, page 9
- `end_fill` .......... project 1b, page 9
- `pu / penup` ....... project 1b, page 10
- `pd / pendown` ... project 1b, page 10
- `hideTurtle` ....... project 1b, page 11
- `setpos` .......... project 1b, page 11
- `pensize` .......... project 1b, page 11

Python

- `from turtle import *` ... project 1a, page 7
- `def` ..................... project 1a, page 7
- `comments` ............ project 1a, page 7
- `pass` ..................... project 1a, page 7
- `return` ................ project 1a, page 7
- `for loop` ............. project 1a, page 7
- `assignment` .......... project 1b, page 12
- `functions` .......... project 1b, page 13
- `function parameters` .. project 1c, page 13
- `default parameters` .... project 1c, page 14