CS1010E Programming Methodology

Semester 2 2019/2020

Assignment 2: Selections and Repetitions

Release Date: 27/01/2020 00:00

Due Date: 02/02/2020 23:59 Total Experience: 100 Exp

Required Files

- `assignment2_template.py`

Notes

- The math library has been imported for you using from `math import *`.
    - If you need any additional library, feel free to add them yourself
    - You are NOT allowed to import turtle in your answer as this may interfere with the test cases on Coursemology.
- You are NOT allowed to change the input arguments to the functions (e.g., *number of arguments*).
- You are NOT allowed to change the function name.
- You may reuse the code written in earlier sub-task for your *subsequent* sub-tasks.
    - You may assume that the correct implementation is given.
    - If you wish to override the standard implementation of earlier sub-task, please copy the code for your earlier sub-task.
    - For instance, if `draw_polygon` calls `draw_square`, you can assume that the correct implementation is given *without* you copying `draw_square` into the answer box.

Task 1: Polygons

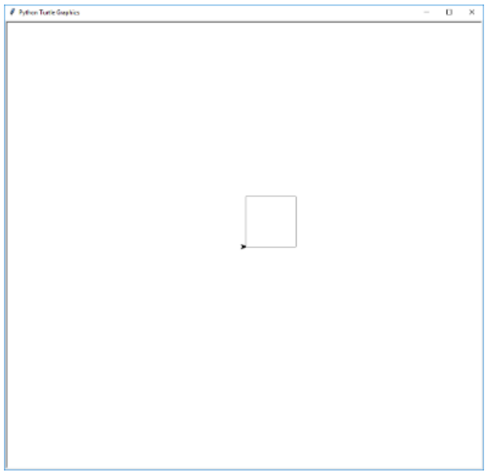
We started learning drawing simple graphics with Python Turtle package in Assignment 1. In this assignment, we will build upon the basic turtle functions to create more sophisticated graphics.

Restrictions

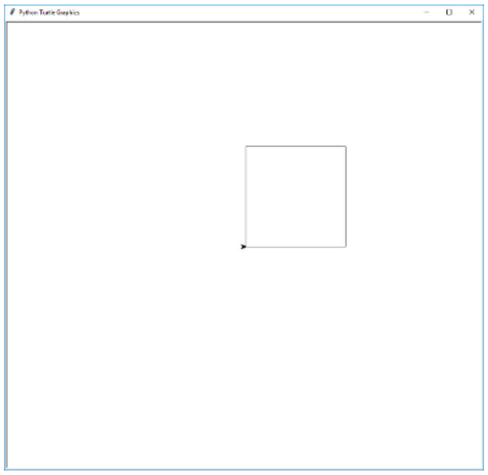
- You are only allowed to use the functions you have learned so far.
    - **`pd`**, **`pu`**, **`fd`**, **`bk`**, **`rt`**, **`lt`**, **`forward`**, **`backward`**, **`right`**, **`left`**.
- You should NOT make any redundant movement. For instance:
    - Going forward, backward, and then forward again.
    - Making left turn 45° followed by another left turn 45° to make a left turn 90°.
- You should NOT lift the pen up (i.e., **`pu()`**).

Task 1A [Warmup]: Drawing a Square [10 Exp]

Write the function `draw_square(d)` that draws a square of length *d*. The start and end point of the rectangle should be at the initial point of the turtle (*bottom-left corner of the square*) as shown in the two images below.



draw_square(100)



draw_square(200)

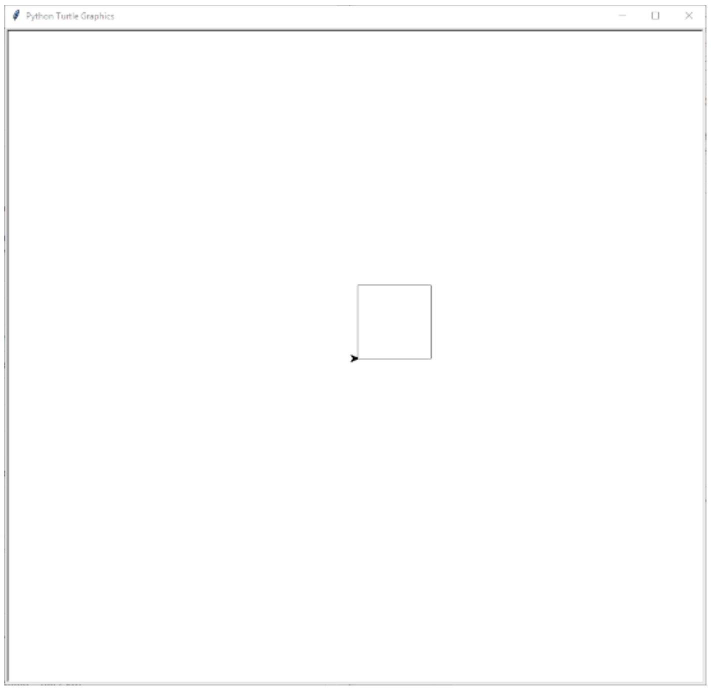
The start and end points in both examples are the same, but the sizes of the squares are different

Assumptions

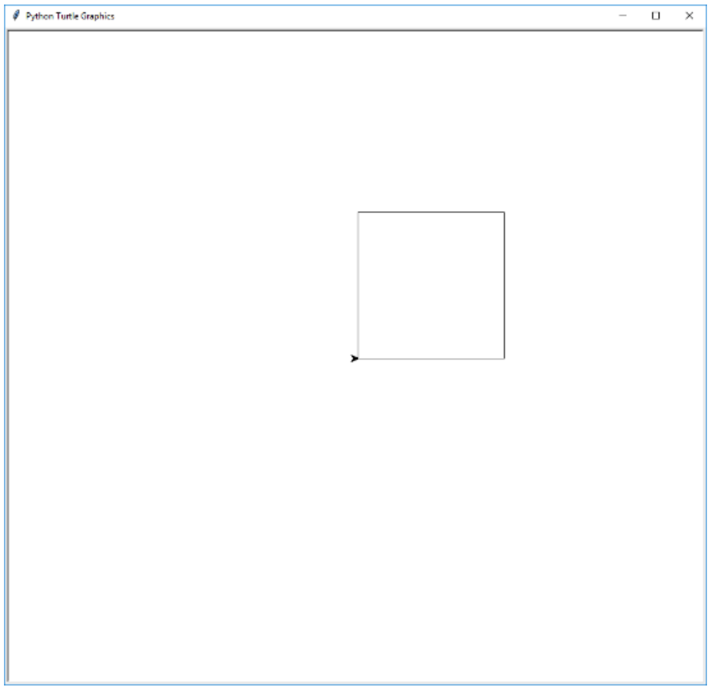
- `d > 0`

Example Run

```
>>> draw_square(100)
```

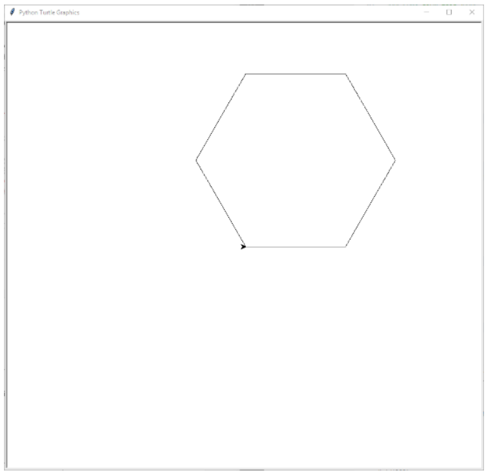


```
>>> draw_square(200)
```

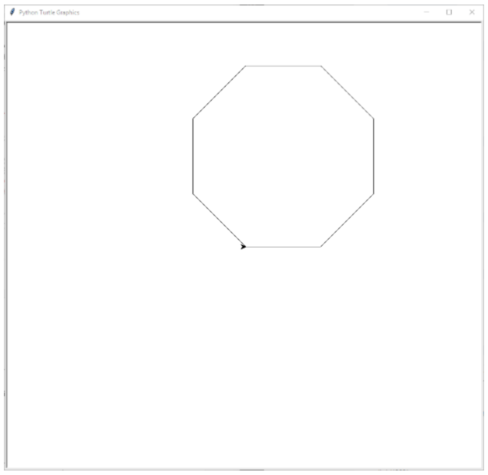


Task 1B: Drawing a Regular Polygon [20 Exp]

Write the function `draw_polygon(d, n)` that draws a regular polygon with *n* sides of length *d*. The regular polygon should be drawn from the initial point of the turtle in a clockwise or an anti-clockwise manner.



`draw_polygon(200, 6)`



`draw_polygon(150, 8)`.

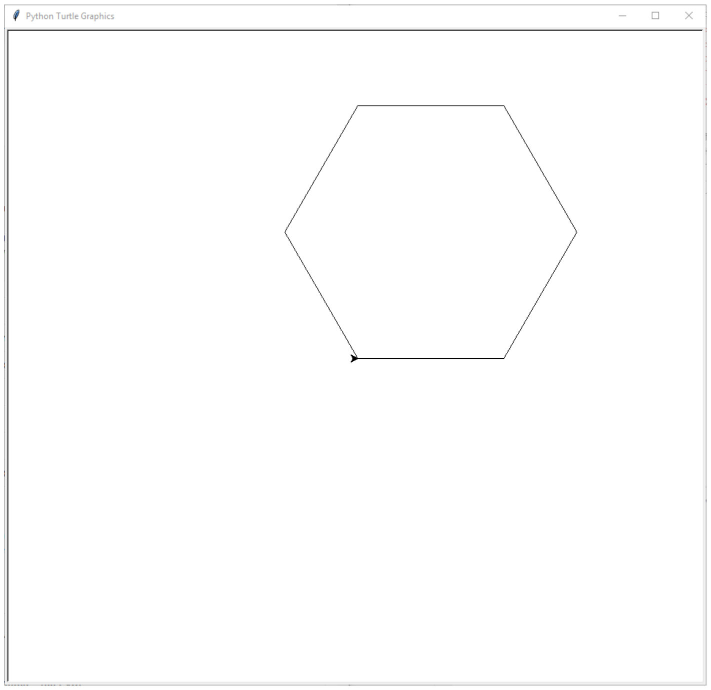
Note: The two examples have different number of sides.

Assumption

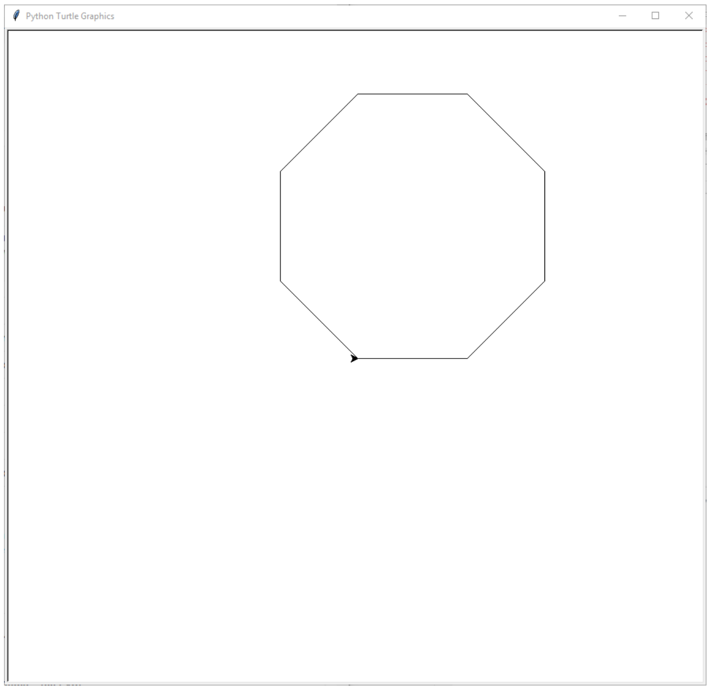
- `d > 0`
- `n >= 3` (*a polygon in 2-dimension must have at least 3 sides*)

Example Run

```
>>> draw_polygon(200,6)
```

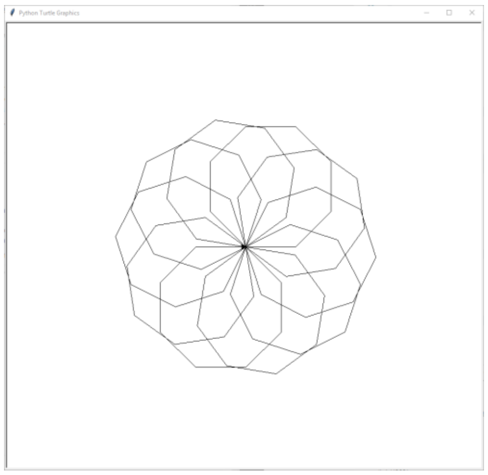


```
>>> draw_polygon(150,8)
```

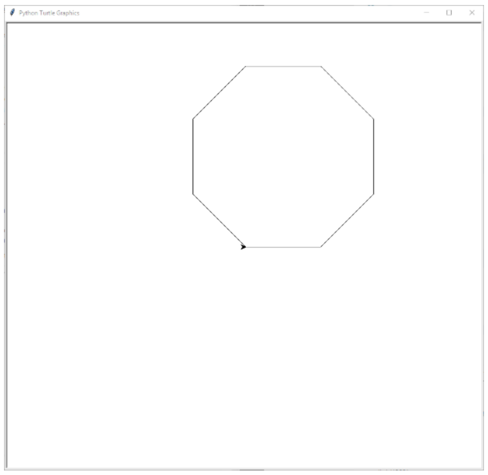


Task 1C: Drawing a Flower [10 Exp]

Write the function `draw_flower(d, n, p)` that draws a flower pattern consisting of `p` regular polygon (*with* `n` *sides and length* `d`) arranged in circular manner. These polygons would be the "petals" of the flower. When `p = 1`, the behavior should be exactly the same as `draw_polygon(d, n)` as shown below.



`draw_flower(100, 8, 10)`



`draw_flower(150, 8, 1)`.

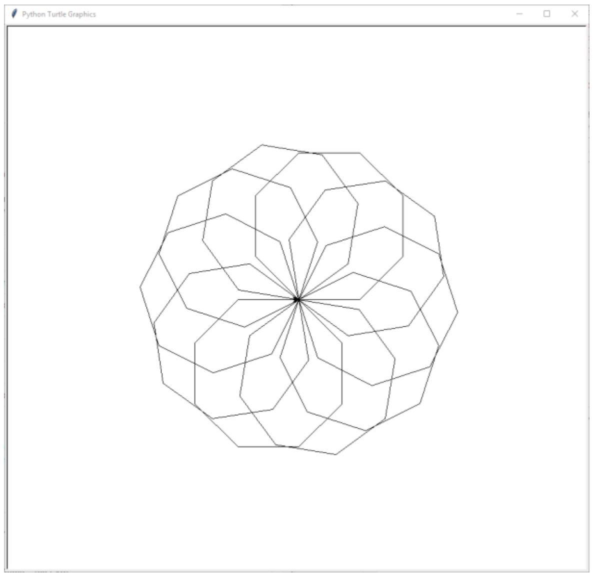
Note: `draw_flower(150, 8, 1)` is the same as `draw_polygon(150, 8)`.

Assumption

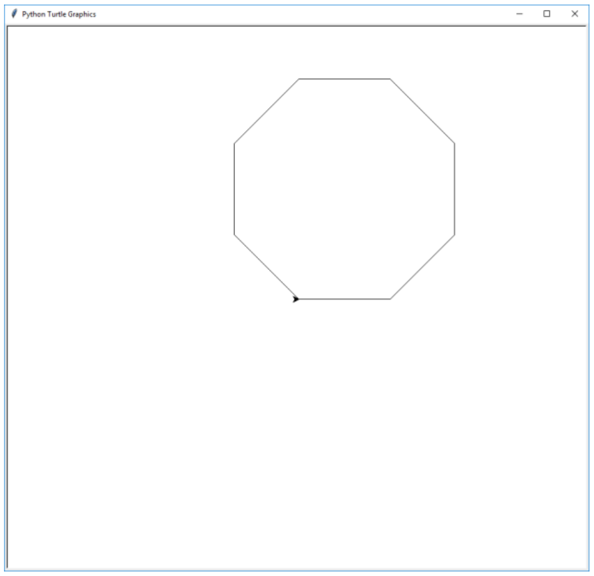
- `d > 0`
- `n >= 3`
- `p >= 1`

Example Run

```
>>> draw_flower(100, 8, 10)
```



```
>>> draw_flower(150, 8, 1)
```



Task 2: Planning for the Future

With selection and repetition statement, you can start planning for your future. In this exercise, we will discuss two important financial plans: loan and retirement.

Restrictions

- You are not allowed to convert any number to any other data type including string.

Task 2A: Loan [20 Exp]

When you take a loan from a bank, they will charge an interest based on the amount of the loan. When you repay the loan yearly, the accrued interest based on the initial amount would be deducted from the repayment and the remainder would be used to repay the loan.

Year	Amount	Payment	Interest	Amount Left
1	$1000	$100	$60	$960
2	$960	$100	$57.4	$917.6
3	$917.6	$100	$55.06	872.66
…	…	…	…	…
16	$68.96	$73.10	$4.14	$0

In this example, you took a loan of $1000 with interest rate of 6% per annum compounded yearly. You decide to repay $100 dollars per year. In the first year, the interest of $60 dollars would be deducted from your repayment. The remaining $40 would be deducted from his loan and the current amount you must pay back is $960. The calculations for the subsequent years are shown on the table above.

Write a function `loan_amount_left(amount, rate, payment, year)` that takes in four inputs: the amount of loan at the start, the annual interest rate, the yearly payment, and the number of years. We assume that the amount is compounded yearly, at the end of the year, after interest is added. Additionally, we assume that the payment is made at the end of the year. The function returns the amount of loan left at the end of the given year.

Assumptions

- `amount >= 0`
- `rate >= 0`
- `year >= 0`

Note

We only check that your final answer is correct up to 3 decimal places, but you should return without rounding.

Example Run

```
>>> loan_amount_left(1000, 6, 100, 1)
960.0

>>> loan_amount_left(1000, 6, 100, 16)
0.0
```

Task 2B: How Long? [10 Exp]

Another interesting question to ask is how long the loan will be completed if you are repaying a certain amount per year. Write a function `loan_length(amount, rate, payment)` that takes in three inputs: the amount of loan at the start, the annual interest rate, and the yearly payment. We assume that the amount is compounded yearly, at the end of the year. Additionally, we assume that the payment is made at the end of the year. The function returns the number of years until the loan is completely paid.

Assumptions

- `amount >= 0`
- `rate >= 0`
- `payment > amount * rate`

The last assumption states that the loan will always be completely paid.

Note

You may assume that the function `loan_amount_left(amount, rate, payment, year)` is already implemented for you correctly based on the assumption above.

Example Run

```
>>> loan_length(1000, 6, 100)
16
```

Task 2C: Retirement [30 marks]

You decided to sign up for a retirement scheme which typically requires yearly deposit into a retirement account. The money deposited into this account would then "grow" in the form of compound interest. The yearly deposit would continue until you are at the retirement age.

After the retirement age, you would instead be withdrawing money from the retirement account based on your preference. This account would still be gathering compound interest, but less due to withdrawal of money.

Let's make this more concrete by using the table of values below. We start with a simple deposit of $10,000 every year for 35 years (assume you start depositing money at 30 years old until you retire at 65 years old) compounded yearly at 3% per annum. The amount of money you will get when you retire is shown below. Assume you make new deposit at the start of the year.

Year	Amount	Interest	Amount at year end
1	$10,000	$300	$10,300
2	$20,300	$609	$20,909
3	$30,909	$927.27	$31,836.27
4	$41,836.27	$1,255.09	$43,091.36
…	…	…	…
35	$604,620.82	$18,138.62	$622,759.44

At the start of your retirement, you are going to have $622,759.44 in your retirement account. Let's say you are going to take $3,000 per month from this retirement account, how long will it take until the retirement account is empty? Let's answer this again with a table! Assume that you are taking at the start of every month, and the interest is given based on your account at the end of the year.

Month	Amount	Withdraw	Interest	Amount at month end
1	$622,759.44	$3,000	$0	$619,759.44
2	$619,759.44	$3,000	$0	$616,759.44
3	$616,759.44	$3,000	$0	$613,759.44
…	…	…	…	…
12	$589,759.44	$3,000	$17,602.78	$604,362.22
13	$604,362.22	$3,000	$0	$601,362.22
…	…	…	…	…
284	$4,716.71	$3,000	$0	$1,716.71
285	$1,716.71	$1,716.71	$0	$0

Note that the amount in both tables are rounded to 2 decimal places for display. However, you should keep the precision as high as possible when computing. Since you withdraw money from the account every month, but the interest is only computed per year, the interest will be non-zero only when the months are multiple of 12. Your last drawn money will be $1,716.71 after 285 months (roughly 23 years and 9 months). It means you can survive with roughly $3,000 a month from your retirement age of 65 until you are 88 years of age.

Write a function `retirement(amount, rate, year, monthly)` that takes in four inputs: the amount of yearly contribution, the annual interest rate compounded yearly, the number of years of contribution, and the monthly payout (i.e., withdrawal) once you reached the retirement age. The function returns the number of months you can get payout (i.e., withdrawal) that are non-zero.

Assumptions

- `amount >= 0`
- `rate >= 0`
- `year >= 0`
- `monthly > amount * rate`

Note

You may assume that the functions `loan_amount_left(amount, rate, payment, year)` and `loan_length(amount, rate, payment)` are already implemented for you correctly based on the assumption above. Hint: what does a negative amount mean?

Example Run

```
>>> retirement(10000, 3, 35, 3000)
285

>>> retirement(10000, 1, 35, 3000)
149
```