Lab 2

COMP9021, Session 2, 2017

1 ☞ Number of trailing 0s in a factorial

To illustrate, 15!, the factorial of 15, is equal to 1307674368000, hence has 3 trailing 0s.

There are at least three methods to compute the number of trailing 0s in the factorial of a number N at least equal to 5:

- Divide $N!$ by 10 for as long as it yields no remainder. Note that for a positive integer `x`, `x // 10` "removes" the rightmost digit from `x`, that digit being equal to `x % 10`.
- Convert $N!$ into a string and find the rightmost occurrence of a character different to `0`. A Google search, or executing `dir(str)` at the python prompt, suggests which string method to use. Note that negative indexes (-1 being the index of the last character in a string, -2 the index of the penultimate character in a string, etc.) is particularly convenient here.
- Python computes such huge numbers as 1000!, either iteratively multiplying all numbers from 1 up to 1000 or using `factorial()` from the `math` module (executing `import math` and then `dir(math)` at the python prompt confirms that this function is available), and the first two methods work for such numbers, but there is a much better method that operates on N rather $N!$, hence that does not suffer the limitations of the first two, and is very efficient. The number of trailing 0s in $N!$ is equal to the number of times $N!$ is a multiple of 10, so to the number of times $N!$ is a multiple of 2×5. It is easy to verify that $N!$ has at least as many multiples of 2 as multiples of 5. Hence the number of trailing 0s in $N!$
  - is equal to the number of times $N!$ is a multiple of 5,
  - which is equal to the number of times 5 occurs in the prime decompositions of 1, 2, ..., $N-1$ and N
  - which is equal to the number of times 5 occurs at least once in the prime decompositions of 1, 2, ..., $N-1$ and N, plus the number of times 5 occurs at least twice in the prime decompositions of 1, 2, ..., $N-1$ and N, plus the number of times 5 occurs at least thrice in the prime decompositions of 1, 2, ..., $N-1$ and N...
  - which is equal to the number of multiples of 5 at most equal to N, plus the number of multiples of 5^2 at most equal to N, plus the number of multiples of 5^3 at most equal to N...

Write a program `trailing_0s.py` that prompts the user for a nonnegative integer N. If the input is incorrect then the program outputs an error message and exits. Otherwise the program computes 5! three times, using the three methods just described.

Here is a possible interaction:

```
$ python3 trailing_0s.py
Input a nonnegative integer: -1
Incorrect input, giving up.
$ python3 trailing_0s.py
Input an integer at least equal to 5: 15
Computing the number of trailing 0s in 15! by dividing by 10 for long enough: 3
Computing the number of trailing 0s in 15! by converting it into a string: 3
Computing the number of trailing 0s in 15! the smart way: 3
$ python3 trailing_0s.py
Input a nonnegative integer: 345
Computing the number of trailing 0s in 345! by dividing by 10 for long enough: 84
Computing the number of trailing 0s in 345! by converting it into a string: 84
Computing the number of trailing 0s in 345! the smart way: 84
$ python3 trailing_0s.py
Input a nonnegative integer: 1000
Computing the number of trailing 0s in 1000! by dividing by 10 for long enough: 249
Computing the number of trailing 0s in 1000! by converting it into a string: 249
Computing the number of trailing 0s in 1000! the smart way: 249
```

2 ☞ Perfect numbers

A number is perfect if it is equal to the sum of its divisors, itself excluded. For instance, the divisors of 28 distinct from 28 are 1, 2, 4, 7 and 14, and $1+2+4+7+14=28$, hence 28 is perfect. Write a program `perfect.py` that prompts the user for an integer N. If the input is incorrect then the program outputs an error message and exits. Otherwise the program outputs all perfect numbers at most equal to N. Implement a naive solution, of quadratic complexity, so that can deal with small values of N only. Here is a possible interaction:

```
$ python3 perfect.py
Input an integer: 100
6 is a perfect number.
28 is a perfect number.
$ python3 perfect.py
Input an integer: 1000
6 is a perfect number.
28 is a perfect number.
496 is a perfect number.
$ python3 perfect.py
Input an integer: 10000
6 is a perfect number.
28 is a perfect number.
496 is a perfect number.
8128 is a perfect number.
```

3 ☞ Decoding a multiplication

Write a program `multiplication.py` that decodes all multiplications of the form

```
   *  *  *
x     *  *
----------
*  *  *  *
*  *  *
----------
*  *  *  *
```

such that the sum of all digits in all 4 columns is constant.

The expected output is:

```
411 * 13 = 5343, all columns adding up to 10.
425 * 23 = 9775, all columns adding up to 18.
```

4 Estimating the probabilities of hypotheses in the light of more and more evidence (moderately advanced, optional practice)

Write a program `bayes.py` that simulates the cast of an unknown die, chosen from a set of 5 dice with 4, 6, 8, 12, and 20 faces. To start with, every die has a probability of 0.2 to be the chosen die. At every cast, the probability of each die being the chosen die is updated using Bayes' rule (find out about it if you do not know it). The probabilities are displayed for at most 5 casts. If more than 5 casts have been requested, the final probabilities obtained for the chosen number of casts are eventually displayed.

Depending on the design of the solution and the way random numbers are generated, different results will be obtained, so we do not use `seed()` as that would not help get outputs that we could reproduce. So the outputs that follow are for illustrative purposes only.

Here is a possible interaction:

```
$ python3 bayes.py
Enter the desired number of times a randomly chosen die will be cast: 2

This is a secret, but the chosen die is the one with 4 faces

Casting the chosen die... Outcome: 4
The updated dice probabilities are:
         4: 37.04%
         6: 24.69%
         8: 18.52%
        12: 12.35%
        20: 7.41%

Casting the chosen die... Outcome: 1
The updated dice probabilities are:
         4: 54.18%
         6: 24.08%
         8: 13.55%
        12: 6.02%
        20: 2.17%
```

```
$ python3 bayes.py
Enter the desired number of times a randomly chosen die will be cast: 5

This is a secret, but the chosen die is the one with 8 faces

Casting the chosen die... Outcome: 7
The updated dice probabilities are:
         4: 0.00%
         6: 0.00%
         8: 48.39%
        12: 32.26%
        20: 19.35%

Casting the chosen die... Outcome: 1
The updated dice probabilities are:
         4: 0.00%
         6: 0.00%
         8: 62.33%
        12: 27.70%
        20: 9.97%

Casting the chosen die... Outcome: 1
The updated dice probabilities are:
         4: 0.00%
         6: 0.00%
         8: 73.51%
        12: 21.78%
        20: 4.70%

Casting the chosen die... Outcome: 3
The updated dice probabilities are:
         4: 0.00%
         6: 0.00%
         8: 81.76%
        12: 16.15%
        20: 2.09%

Casting the chosen die... Outcome: 7
The updated dice probabilities are:
         4: 0.00%
         6: 0.00%
         8: 87.57%
        12: 11.53%
        20: 0.90%
```

```
$ python3 bayes.py
Enter the desired number of times a randomly chosen die will be cast: 20

This is a secret, but the chosen die is the one with 6 faces

Casting the chosen die... Outcome: 1
The updated dice probabilities are:
         4: 37.04%
         6: 24.69%
         8: 18.52%
        12: 12.35%
        20: 7.41%

Casting the chosen die... Outcome: 4
The updated dice probabilities are:
         4: 54.18%
         6: 24.08%
         8: 13.55%
        12: 6.02%
        20: 2.17%

Casting the chosen die... Outcome: 6
The updated dice probabilities are:
         4: 0.00%
         6: 63.54%
         8: 26.80%
        12: 7.94%
        20: 1.72%

Casting the chosen die... Outcome: 6
The updated dice probabilities are:
         4: 0.00%
         6: 72.10%
         8: 22.81%
        12: 4.51%
        20: 0.58%

Casting the chosen die... Outcome: 5
The updated dice probabilities are:
         4: 0.00%
         6: 78.68%
         8: 18.67%
        12: 2.46%
        20: 0.19%

The final probabilities are:
         4: 0.00%
         6: 99.68%
         8: 0.32%
        12: 0.00%
        20: 0.00%
```

5 Drawing hypotrochoids and epitrochoids with turtle (advanced, optional practice)

Study the notes on cycloidal curves, and write a program `cycloidal_curves.py` that uses `turtle` to prompt the user to—check out `turtle.textinput()` and `turtle.numinput()`:

- choose between drawing either an epitrochoid (providing no input) or a hypotrochoid (providing any input),
- input the radius R of the fixed circle, indicating min and max admissible values,
- input the radius r of the rolling circle, indicating min and max admissible values,
- input the distance between the drawing point and the centre of the rolling circle, indicating min and max admissible values,

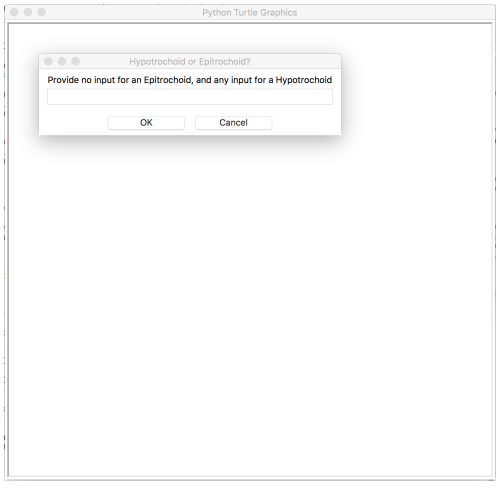
and then draws the chosen hypotrochoid or epitrochoid.

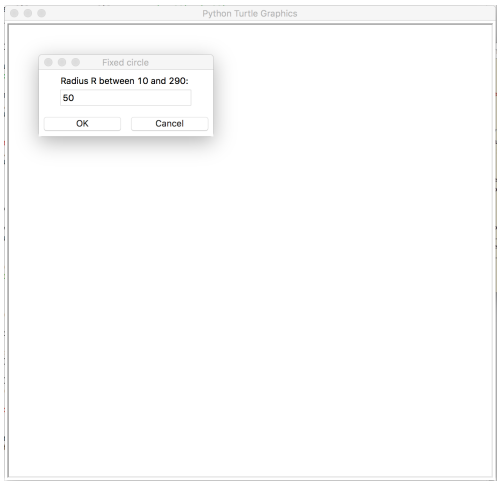
Denoting by *dim* the size of the window—check out `turtle.screensize()`—, we want:

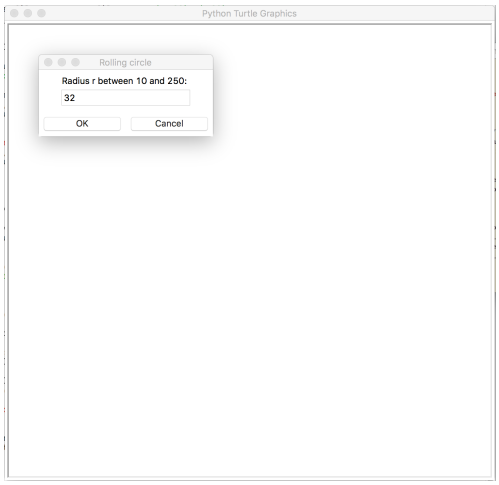
- R to be between 10 and $dim - 10$,
- r to be at least equal to 10 and such that the centre of the rolling circle is always at most *dim* away from the origin,
- d to be at least equal to 0 and such that the drawing point is at most *dim* away from the origin,
- epitrochoids to be filled in green and hypotrochoids in yellow,
- the top of the window to display—check out `turtle.title()`—one of:

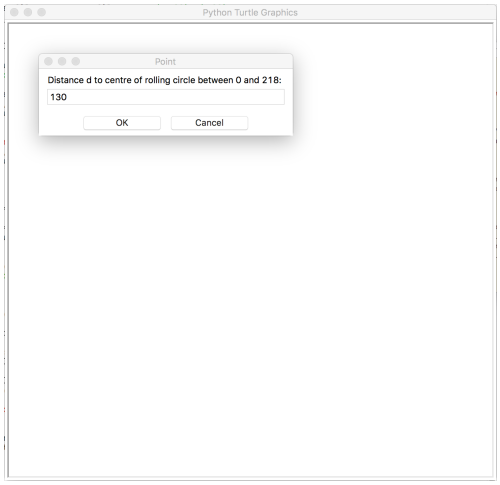
```
Epitrochoid for R = ..., r = ..., d = ... -- Period = ...
Hypotrochoid for R = ..., r = ..., d = ... -- Period = ...
```

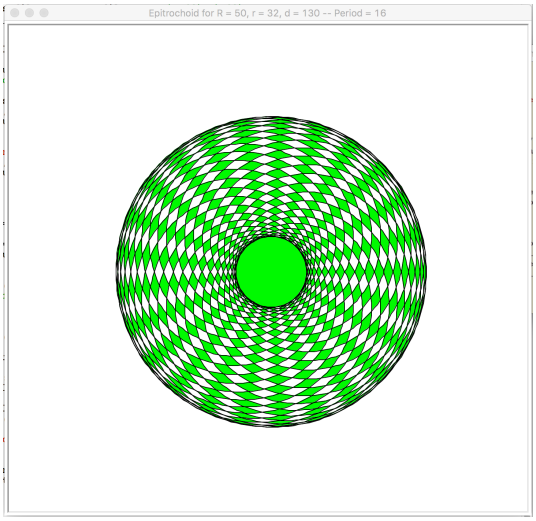
Here are screenshots of a possible interaction:











It is suggested to draw the curve moving by an angle of one degree from one point to the next.

The solution that will be provided will give the user the option to save the drawing as a postcript file—check out `turtle.getcanvas().postscript()`—after the picture has been drawn by pressing the `S` key—check out `turtle.listen()` and `turtle.onkey()`—, and using the `PIL` module, open this postcript file—check out `PIL.Image.open()`—and save it as a pdf—check out `PIL.Image.Image.save()`, naming it `Epitrochoid_R_r_d.pdf` or `Hypotrochoid_R_r_d.pdf`, before deleting the postcript file—check out the `remove()` function from the `os` module.