Python Exercises

Contents

1 Beginner

1.1 Hi, guys!

Define a function called `greet`, which takes no argument and prints a random string by choosing a random greeting response from one of {*"Welcome :))"*, *"Heyy!"*, *"Sup bro!"*, *"Ahoy!"*, *"Howdy-doody"*, *"Greetings and salutations!"*, *"Yo!"*, *"Hola!"*, *"Konichiwa!"*, *"There you are!"*, *"Well well, look at you!"*, *"Salute plurimam dicit. Si vales, bene est, ego valeo."*, *"Valar morghulis"*. }

Hints:

1. `random.choice`

1.2 PrintPrint..Print $nn..n$ timestimes..times

Define a function called `print_n_times`, which takes two arguments, a string `s` and an integer `n`, and prints the string `n` times, each on a separate line. If `n` is meaningless (negative, float etc.) or `s` is not string whatsoever, then call `greet` function from 1.1.

Hints:

1. If `greet` is far far away, `import` it.

2 Intermediate

2.1 Heads Tails Heads Tails Heads Tails

The coin flip experiment results in binomial distribution. The probability of getting exactly k successes in n trials is given by

$$\binom{n}{k} p^k (1-p)^{n-k} \tag{1}$$

where $p = 0.5$ is the probability of success in one event (heads or tails) and

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2}$$

Define a function called `get_coin_flip_prob` that takes n and k as arguments and returns the probability in Equation 1.

Hints:

1. `math.factorial`

2.1.1 Sample size matters

Test your function by computing the probability of 51 tails out of 100 trials and 510 tails out of 1000 trials. Which one is more probable?

2.2 Am I primal?

Write a function that checks given number is prime or not. If it is, print *"Congrats! I am proud, you are prime."*; if not, print *"Well, you are not primal, but highly evolved."*

Hints:

1. Go up to square root.

2.3 Long live Euclid!

Euclid' s algorithm is a method for finding the greatest common divisor (GCD) of two numbers. Recall that the GCD of two numbers `m` and `n` is the largest number that divides both m and n.

Algorithm 1 Division method

```
function GCD(a, b)
    while b ≠ 0 do
        temp = b
        b = a mod b
        a = temp
    end while
    return a
end function
```

Algorithm 2 Subtraction method

```
function GCD(a, b)
    while a ≠ b do
        if a > b then
            a = a - b
        else
            b = b - a
        end if
    end while
    return a
end function
```

Algorithm 3 Recursive method

```
function GCD(a, b)
    if b = 0 then
        return a
    else
        return GCD(b, a mod b)
    end if
end function
```

2.3.1 Divide, subtract and recurse

Implement all three versions above which takes two integers as its arguments and returns their GCD. You may assume that both inputs are integers, so there is no need to include any error checking in your function.

2.3.2 Dare testing Euclid?

Test your functions for following pairs and compare functions to their running times.

- 13, 13
- 20, 10
- 2017, 2018
- 5040, 60
- 73, 37

2.4 B$e$hold, $e$ul$e$r!

The base of the natural logarithm, e, can be defined as the infinite sum (Taylor expansion)

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \dots \tag{3}$$

But earlier (18th century), Bernoulli defined the compound interest problem

$$\lim_{x \to \infty} \left(1 + \frac{1}{x}\right)^x \tag{4}$$

Suppose you have a money in the bank. And the bank compounds the money annually. Equation 4 gives the increase rate of money if 100% interest is compounded in one year where x defines the number of the time that interest is applied. If $x = 1$, 100% interest is applied only at the end of the year. Then total rate becomes 2, namely money is doubled. If $x = 2$, 50% interest is applied twice through one year. In this case, total rate becomes 2.25. What happens when $x \to \infty$ as in the Equation 4? This is where Euler comes in. He calculated this converging sum and ended up an irrational number $e = 2.718281...$

2.4.1 Comput$e$ $\lim_{x \to \infty}(1 + \frac{1}{x})^x$ ul$e$r

Define a function called `euler_limit` that takes number n and approximates Equation 4. You may assume that the input to your function will be a positive integer.

2.4.2 Comput$e$ $\sum_{k=0}^{\infty} \frac{1}{k!}$ ul$e$r

Define a function called `euler_infinite_sum` that takes a non-negative integer argument n, and returns an approximation to e based on the first n terms of Equation 3.

Hints:

1. `euler_infinite_sum(0)` should give 0.

2.4.3 Get to know ϵul$e$r?

Define a function called `euler_approx` that takes an argument, a float *epsilon*, and uses the sum in Equation 3 for an approximation of e that is within *epsilon* of the true value of e.

Hints:

1. `math.e`

2.4.4 Say his name rep$e$at$e$dly!

Define a function called `print_euler_sum_table` that takes a positive integer n as an argument and prints the successive values obtained from `euler_infinite_sum(k)` as k ranges from 1 to n, one per line.

3 Advanced

3.1 A Story of Creation

3.1.1 What are Gods? Immortal Men.

In the beginning, there was only Chaos. No space, no time just endless Chaos. But Chaos was made of two siblings: the darkness (Erebus) and the night (Nyx). Their love gave birth to the Mother Earth, Gaia. And Gaia gave birth to the Father Sky, Uranus to equally cover herself. Then from their love 6 Titans, 6 Titanesses, one-eyed Cyclopses and hundred-arms Hecatoncheires were born. Titans ruled the Universe for a very long time.
Neither Father Uranus did not love his children nor the children did not love him back. He imprisoned them in Tartarus deep within the Earth where they gave pain to Gaia. She incited a riot among her children. One of the children, Cronus was willing to help his mother. He castrated Uranus and threw his testicles into the sea from which Aphrodite, the Goddess of Love was emerged.
New ruler, Cronus and his sister Rhea gave birth to the Hestia, Demeter, Hera, Hades, Poseidon and Zeus. Cronus thought that Zeus would be an opponent against him so he wanted to eat Zeus up. Rhea deceived Cronus by giving him a stone wrapped in baby's clothes.
After years, Gods have declared a war against the Titans. Gods have prevailed. Poseidon, Hades and Zeus possessed the Sea, the Underworld and the Sky respectively and sat on the throne in Mount Olympus. The Age of Gods had begun.

Define an `Immortal` class with following attributes: `name, gender, essence`.

- `name` : String. Name of the immortal.
- `gender` : String. Gender of the immortal. `female` or `male`.
- `essence` : String. Origin of the immortal. e.g. *Sky* for *the God of Sky.*

Write an initializer method that takes `name, gender, essence`. Immortals are binary(!) (`gender` must be `female` or `male`). Check if gender is given correctly.

Hints:

1. `assert`

3.1.2 Immortals, how should I *call* you?

Define two classes that inherit from `Immortal`: `Titan` and `God`. Both have attributes `name, gender, essence`. Write printing methods for both `Titan` and `God` to see the name and essence when you print it. e.g. *"Apollo, the God of Sun"*, *"Athena, the Goddess of Wisdom"*, *"Gaia, the Titaness of Earth"*.

Hints:

1. `__str__` or `__repr__`
2. Mind the gender!
3. Name and essence are titled.

3.1.3 What are Men? Mortal Gods.

…Titans were imprisoned in the eternal hell of Tartarus. But two Titans, Prometheus and Epimetheus stood by the side of the Gods. Zeus honored them with the task of creating all living things on Earth. Epimetheus gave the creatures a portion of Gods' abilities. Some had the ability to swim, to fly or to run fast. Some had thick furs and sharp claws for hunting. Meanwhile, Prometheus created humans with a great effort and gave them a reflection of the image of Gods. Zeus worried of seeing their reflections from above. He warned Prometheus that Humans must remain mortal and worship the Gods themselves.

Although humans have specialized in many areas like craft, art, philosophy, science, they were in need of protection and care of the Gods to stay alive on the wild Earth.

Define a `Mortal` class with following attributes: `name, gender, profession`.

- `name` : String. Name of the mortal.
- `gender` : String. Gender of the mortal.
- `profession` : Optional String. Default value is empty string. Profession of the mortal.

Write an initializer method that takes `name`, `gender` and optional `profession`. Mortals are free in genders. (Any type of gender is acceptable, no need `Assertion`).

Define classes that inherit from `Mortal`: `Human` and `Creature`. All have attributes `name, gender, profession`. Write a printing method for all to see the name and the profession of the mortal when you print it. e.g. *"Mortal man Achilles the Warrior"*, *"Mortal man Homer the Author"*, *"Mortal woman Hypatia the Mathematician"*, *"Mortal creature Succubus the Seducer"*.

3.1.4 Prometheus, show me how to *burn*!

...Prometheus was in agony to see his creations under oppression. He felt anger and vengeance against Zeus. So he presented less valuable parts of the meat of sacrifices made by Humans to the Gods. This made Zeus mad. He forbade the use of Fire on Earth to cook the meat or for any other purposes.
But Prometheus could not bear that his creations were unable to use this power. He climbed to Mount Olympus where the Fire was hidden inside the forge of Hephaestus, the God of Fire. He carried it down to the Humans. This power released the potential of Human mind and imagination. Hereafter, humans were able to harness nature, obtain better food resources and forge weapons. Civilization progress ramped up.
Zeus, the God of Gods were outraged when he saw all the lights spreaded around the Earth. He ordered that Prometheus was to be chained at the top of Mount Caucasus. A vulture would come daily to eat his liver which will be replenished at the end of the day. Prometheus was captured in an endless painful loop. However, he was at peace when looked at down below and see the sea of lights that illuminate his creations.

Define `_illumination` variable for only `Human` class at its initializer. Write `get` and `_set` methods for it. Define `illuminate` method that calls `_set` to increment `_illumination`. All the changes defined below in detail should be in `Human` class. Remember, Humans got the Fire!

- `_illumination` : Private Integer. At initializer. Initial value is 1 (A faint flame).
- `_max_illumination` : Private Integer. At initializer. Initial value is 1000.
- `get_illumination()` : Public method. Returns `_illumination`.
- `_set_illumination(n)` : Private method. Sets `_illumination` to `n`. Bound it between [0, `_max_illumination`]. (**Hint** : max() of min() or min() of max())
- `illuminate(delta)` : Public method. Increments `_illumination` by `delta` calling `_set_illumination(n)` and `print` new `_illumination` value e.g. *Archimedes's illumination is 73.* Prevent exceeding `_max_illumination`.

3.1.5 Eternal Fire, show me how to become Immortal!

"What you leave behind is not what is engraved in stone monuments, but what is woven into the lives of others." — Pericles

Define `influence_range` for all classes that you have written. Define `influence(character)` method for both `Immortal` and `Mortal` classes. Define `_become_immortal()` method for only `Human` class. When `_max_illumination` is reached at `illuminate(delta)` method call `_become_immortal()`.

- `influence_range` : Public `range`. At initializer of all classes. Initial values are as follows:
  - `range(-1, 2)` for `Immortal`
  - `range(-1, 2)` for `Mortal`
  - `range(-2, 7)` for `Titan`
  - `range(-5, 9)` for `God`
  - `range(-1, 11)` for `Human`
  - `range(-3, 5)` for `Creature`
- `influence(character)` : Public method. At only `Immortal` and `Mortal` classes. Takes a `Human` character, chooses random number `delta` in `influence_range` and call `illuminate(delta)` method. (**Hint**: Check character type with `assert`.)
- `_become_immortal()` : Private method. At only `Human` class. First check if `_illumination` reached to `_max_illumination`. Then change class type to `Immortal`, assign `profession` to `essence` and `print` new `self`. Call `_become_immortal()` method under `illuminate(delta)` method when `_max_illumination` is reached. (**Hints**: Use `__class__` to change class type. Assign `self.profession` to a variable before changing class type.)

3.1.6 Ignite, burn and explode!

...The Fire was spread on Earth at the hands of Humans who are blessed and guided by Prometheus, the Titan of Fire. One of the regions on Earth where the lights of Fire shine very bright was called Ionia. Ionia's people held the Fire for generations and they harnessed the power of it to turned into knowledge and imagination. And some of the people were illuminated brightest among the mortals. So bright that meant to never fading fire which would be burning to the end of time, eternity. Their Fire guided all others who wanted to escape from darkness as once Prometheus guided them. They had become Immortals.

Define a stand-alone function (not a part of any class) `spread_the_fire(influencers, flame_holders)` as described below.

- `influencers` : List of `Immortal` and `Mortal` instances.
- `flame_holders` : List of `Immortal` and `Mortal` instances. Both list should contain at least one `Human`. The best influencers for Humans are Humans themselves! (**Hint** : Check with `any` and `assert`)

Algorithm 4 *Spreading the fire* algorithm

```
function SPREAD_THE_FIRE(influencers, flame_holders)
    assert flame_holders contain at least one Human
    assert influencers contain at least one Human
    while True do
        Randomly choose one influencer
        Randomly choose one flame_holder
        try influencing flame_holder
        if AssertionError is caught then
            print the error
            continue
        end if
        if flame_holder became an Immortal then
            break
        end if
    end while
end function
```

Some `Immortal` and `Mortal` instances you might want to use are as follows:

• *Gaia, the Titaness of Earth* • *Uranus, the Titan of Sky* • *Chronus, the Titan of Harvest* • *Atlas, the Titan of Endurance* • *Prometheus, the Titan of Fire*
• *Zeus, the God of Sky* • *Poseidon, the God of Sea* • *Hades, the God of Underworld* • *Apollo, the God of Sun* • *Athena, the Goddess of Wisdom* • *Dionysus, the Goddess of Wine* • *Aphrodite, the Goddess of Love* • *Ares, the God of War* • *Hera, the Goddess of Marriage* • *Artemis, the Goddess of Hunt* • *Demeter, the Goddess of Harvest*
• *Mortal man Homeros the Author* • *Mortal man Thales the Philosopher* • *Mortal man Socrates the Philosopher* • *Mortal man Plato the Philosopher* • *Mortal man Aristotle the Philosopher* • *Mortal man Archimedes the Inventor* • *Mortal man Aristarchus the Astronomer* • *Mortal man Herodotus the Historian* • *Mortal woman Theano the Philosopher* • *Mortal man Sophocles the Poet* • *Mortal man Ictinus the Architect* • *Mortal man Phidias the Sculptor* • *Mortal woman Hypatia the Mathematician*
• *Mortal gorgon Medusa the Monster* • *Mortal horse Pegasus the Flyer* • *Mortal hound Cerberus the Guard*

Using the algorithm and sample characters described above, run
`spread_the_fire(influencers, flame_holders)` and see who will be the first `Immortal`.
You can also define an `_influencers` variable to see the most influent characters upon your `Immortal`.

- `_influencers` : Dictionary(`Immortal` or `Mortal`, Integer). Keys are influencer instance, values are their influence amount.
- `get_influencers()` : Public method. Returns `_influencers`.
- `_set_influencers(influencer, delta)` : Private method. Increments the value of `influencer` in `_influencers` dictionary by `delta`.

Hints:

1. Call `_set_influencers(influencer, delta)` in `Human` class. Do not call it from outside!
2. Use `setdefault` to set default value of `_influencers` dictionary.
3. Get `influencer` object with `inspect.currentframe().f_back.f_locals['self']`
4. `import inspect`
5. Set new `influencers` dictionary for Mortals that became Immortals since you cannot reach `get_influencers()` method.

3.2 Again, $euler$

Euler method is used to solve ordinary differential equations (ODE)s.

General first order differential equation form is

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0 \tag{5}$$

We want to approximate the solution to (1) near t_0. We only know the value of the solution and its derivative at initial point. We can get this by plugging the initial condition into $f(t, y)$ into the differential equation itself. So, the derivative at this point is.

$$\left.\frac{dy}{dt}\right|_{t=t_0} = f(t_0, y_0) \approx \frac{y - y_0}{t - t_0} \tag{6}$$

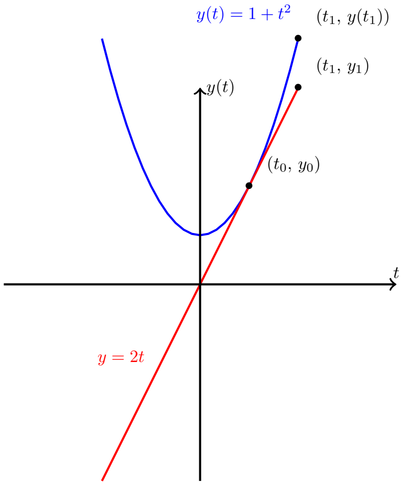

Figure 1: Sample approach for the plot of $f(x) = 1 + t^2$ with a tangent at $t_0 = 1.0$.

In Fig. 1, if t_1 is close enough to t_0 then the point y_1 on the tangent line should be fairly close to the actual value of the solution at t_1. Finding y_1 is easy enough. All we need to do is plug t_1 in the equation for the tangent line.

$$y_1 = y_0 + f(t_0, y_0)(t_1 - t_0) \tag{7}$$

This y_1 is only an approximation. If we accept the error, we can continue build up lines similarly.

$$y_2 = y_1 + f(t_1, y_1)(t_2 - t_1) \tag{8}$$

$$y_3 = y_2 + f(t_2, y_2)(t_3 - t_2) \tag{9}$$

In general, starting from the initial point, (t_n, y_n) we have an approximation at (t_{n+1}, y_{n+1}) like

$$y_{n+1} = y_n + f(t_n, y_n)(t_{n+1} - t_n) \tag{10}$$

If we were to take a constant step size

$$t_{n+1} - t_n = h \tag{11}$$

Equation (6) would be,

$$y_{n+1} = y_n + h.f(t_n, y_n) \tag{12}$$

But of course this step size may not be constant, but adaptive.

We start at initial point (t_0, y_0) and repeatedly evaluate new points with selected or adaptive step size. We continue until we reach the target point that we want to compute (We will acquire $y(t_f)$ for target point t_f).

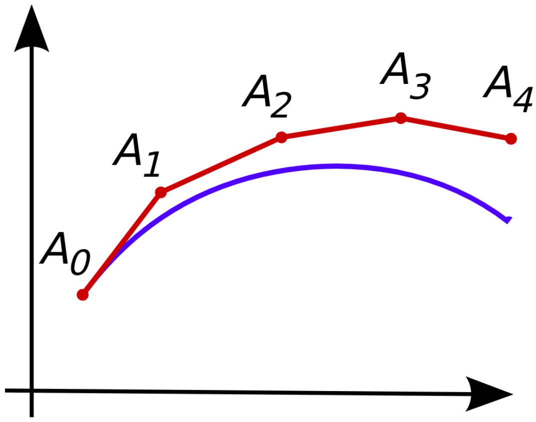

Figure 2: Euler approximation steps

Here is a pseudo-code for algorithm where n is the number of steps.

Algorithm 5 Euler method with constant step size

function EULER(f, t_0, y_0, n, t_f)
    Compute step size h
    for i from 1 to n **do**
        Compute $f(t_0, y_0)$
        Compute new t_1
        Compute corresponding y_1
        Update t_0 and y_0
    end for
    return y_0
end function

3.2.1 Patience! Step by step

Write a function that takes function f, initial points t_0 and y_0, number of steps n and target point t_f and returns approximate y_f value.

3.2.2 Dare to test euler?

Test your function for following differential equations with given initial points at desired target points with different number of steps n. Compare your results to given analytic solutions. Show your numerical results and analytic results and their relative errors for different n' s.

$$i) \quad \frac{dy}{dt} = y + t \qquad y(0) = 0 \qquad\qquad \textbf{analytic solution:}(y = e^t - t - 1)$$

$$ii) \quad \frac{dy}{dt} = \sin t - \frac{y}{t} \qquad y(0) = 0 \qquad\qquad \textbf{analytic solution:}(y = \frac{\sin t}{t} - \cos t)$$

Hints:

1. Use only `math` library.
2. relative error $= \frac{|\text{predicted value} - \text{true value}|}{\text{true value}}$

3.3 Famous Monte Carlo

Monte Carlo methods are a broad class of computational algorithms based on repeated random sampling used for making numerical computations.

You will implement such an experiment to compute approximate value of π.

In Monte Carlo simulations two basic sampling methods exist. One of them is *direct sampling.* In the direct sampling method, we take samples independently for each step by choosing a random position in the space. Hence, if we use real random numbers, our new samples are independent of the previous ones, because we choose a new position randomly. By taking samples again and again, we sweep out the space and by using these samples, we can make calculations.

Other sampling method is *Markov-Chain sampling.* In the Markov-Chain sampling, our sample depends on the previous sample. Firstly, we start sampling from an initial position that is given or where the last simulation ends. Then, from the initial site, we move to another site on the space, in any direction and distance. This direction and distance again random, but, distance is limited to a value δ. By using this procedure, we visit other states and if the resulting state out of our space, then we reject the move. Hence, this limitation for the moves δ affects the rejection rate. If δ is very large, rejection rate shold be too high, this means that we generally do not move to another state; thus, our traveled path is small. Also, if δ is very small, acceptance rate becomes too high, this means that we generally move from state to another state; however, in this case our traveled path is, again, small and we cannot sweep out the space.

In this question you select random points with these two methods in space with circle and square where r is the radius of the circle and $2r$ is the length of square as seen below.

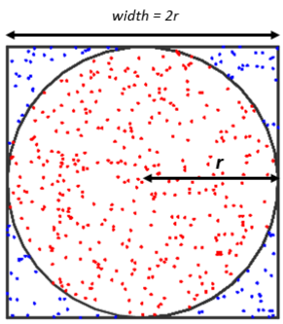

Figure 3: Collection of random selected points in sample space (tangential circle in a square)

3.3.1 Reπresent with πower and reπetition

Use *direct sampling* to simulate experiment and compute approximate π value. Your simulations should take two parameters: *power* and *repetition.* Power represents number of samples. For example if power = 5, then number of samples should be 10^5. *Repetition* represents number of repetition of one simulation to take the average of the π value. For example run your simulation with *repetition* = 10 and take the average π for those 10 simulations. Compare your results to real π value for different *power* and *repetition* values.

3.3.2 Chained to Markov!

Use *Markov-chain sampling* to simulate experiment and compute approximate π value. Start from random point in the space and move by random δ_x and δ_y along each axis at each step. If you go out of space, reject that point and choose different random points. Evaluate π values and rejection rate by keeping step size constant and changing number of samples; and keeping number of samples constant and changing step size.

Hints:

1. Use only `math` and `random` libraries.
2. Use circle-to-square area ratio and solve for π.