Assignment 0 Workshop

- Jonas Raaschou-Pedersen

Today's plan:

1. Go through the questions asked for this workshop
2. Go through any questions asked during the workshop

Question 1:

”I am a bit unsure about the concept of nested functions. Particularly, the part of the assignment that says:

> *'The nested function should return func(e, k) where e=eulers_e(x, k)'."*

Problem 0.3.5

Just like numbers, strings and data types, python treats functions as an object. This means you can write functions, that take a function as input and functions that return functions after being executed. This is sometimes a useful tool to have when you need to add extra functionality to an already existing function, or if you need to write *function factories*.

> **Ex. 0.3.5:** Write a function called `exponentiate` that takes one input named `func`. In the body of `exponentiate` define a nested function (i.e. a function within a function) called `with_exp` that takes two inputs `x` and `k`. The nested function should return `func(e, k)` where `e=eulers_e(x, k)`. The outer function should return the nested function `with_exp`, i.e. write something like

```
def exponentiate(func):
    def with_exp(x, k):
        e = eulers_e(x, k)
        value = #[FILL IN]
        return value
    return with_exp
```

> Call the `exponentiate` function on `natural_logarithm` and store the result in a new variable called `logexp`. Then call `logexp(1, 1000)` and store this value in the variable `answer_035`.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```
assert round(answer_035) == 1
assert round(logexp(2, 1000)) == 2
assert round(logexp(3, 1000)) == 3
assert round(logexp(4, 1000)) == 4
```

Nested functions

A nested function is a function defined inside another function.

Let's try to make a function that uplifts a number, `x`, to an given exponent, `exponent`.

The function is defined below.

In [69]:

```
def uplift_to(exponent):
    """
    returns the value uplifted to a power
    """
    def uplifted(x):
        return x ** exponent
    return uplifted
```

Let's try to apply the function on the number, x=2 .

In [70]:
```
#Defined functions using the concept of a nested function.
quadratic = uplift_to(2)
cubic = uplift_to(3)
quartic = uplift_to(4)
quintic = uplift_to(5)

#list of the functions defined above
list_of_functions = [quadratic, cubic, quartic, quintic]
function_names = ["Quadratic", "Cubic", "Quartic", "Quintic"]

#Loop through the two lists above, print out the name of the function
# and the value of the funciton evaluated at x = 2
for name, function in zip(function_names, list_of_functions):
    print(f"Function: {name}. Value of function evaluated at x = 2: {function(2)}")
```

```
Function: Quadratic. Value of function evaluated at x = 2: 4
Function: Cubic. Value of function evaluated at x = 2: 8
Function: Quartic. Value of function evaluated at x = 2: 16
Function: Quintic. Value of function evaluated at x = 2: 32
```

Returning to Problem 0.3.5.

The problem basically asks you to define a function, which spits out a function evaluated at `e = eulers_e(x, k)`.

In the case of the natural logarithm, the functions spits out $\log(\exp(x))$ for the approximation that uses the value k.

Using the fact that $\log(\exp(x)) = x$, we expect that the approximations are close to x, thus the assert statements.

Problem 0.3.5

Just like numbers, strings and data types, python treats functions as an object. This means you can write functions, that take a function as input and functions that return functions after being executed. This is sometimes a useful tool to have when you need to add extra functionality to an already existing function, or if you need to write *function factories*.

> **Ex. 0.3.5:** Write a function called `exponentiate` that takes one input named `func`. In the body of `exponentiate` define a nested function (i.e. a function within a function) called `with_exp` that takes two inputs `x` and `k`. The nested function should return `func(e, k)` where `e=eulers_e(x, k)`. The outer function should return the nested function `with_exp`, i.e. write something like

```
def exponentiate(func):
    def with_exp(x, k):
        e = eulers_e(x, k)
        value = #[FILL IN]
        return value
    return with_exp
```

> Call the `exponentiate` function on `natural_logarithm` and store the result in a new variable called `logexp`. Then call `logexp(1, 1000)` and store this value in the variable `answer_035`.

In [ ]:
```
# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```
assert round(answer_035) == 1
assert round(logexp(2, 1000)) == 2
assert round(logexp(3, 1000)) == 3
assert round(logexp(4, 1000)) == 4
```

In [72]:
```
def exponentiate(func):
    def with_exp(x, k):
        e = eulers_e(x, k)
        value = func(e, k)
        return value
    return with_exp
```

Using the function above, apply it to the `natural_logarithm()` and see if the correct answer comes out.

Note that the `natural_logarithm()` and `eulers_e()` need to be correct for the nested function to work correctly :)

Question 2:

Getting an overflow error when applying my natural_logarithm() function at the x=e, where e is eulers number.

What is happening in the code below?

In [66]:
```
def nat_log(x, k_max):
    total = 0
    for k in range (0, k_max+1):
        exp = 2*k+1
        total = total + 1/(exp)*((x-1)/(x+1))**(exp)
        total = total*2
    return total
nat_log(2.71, 10000)
```

Out[66]:
```
inf
```

The function does not compute the equation below

$$log(x) = 2 \cdot \sum_{k=0}^{\infty} \frac{1}{2k+1} \left(\frac{x-1}{x+1} \right)^{2k+1}$$

Let us try to modify the code on the previous slide.

In [71]:
```
def nat_log(x, k_max):
    total = 0
    for k in range (0, k_max+1):
        exp = 2*k+1
        val = 2*(1/(exp)*((x-1)/(x+1))**(exp))
        total +=val
    return total
nat_log(2.71, 1000)
```

Out[71]:
```
0.9969486348916091
```

The function now computes the approximation in the equation below.

$$log(x) = 2 \cdot \sum_{k=0}^{\infty} \frac{1}{2k+1} \left(\frac{x-1}{x+1} \right)^{2k+1}$$

Question 3:

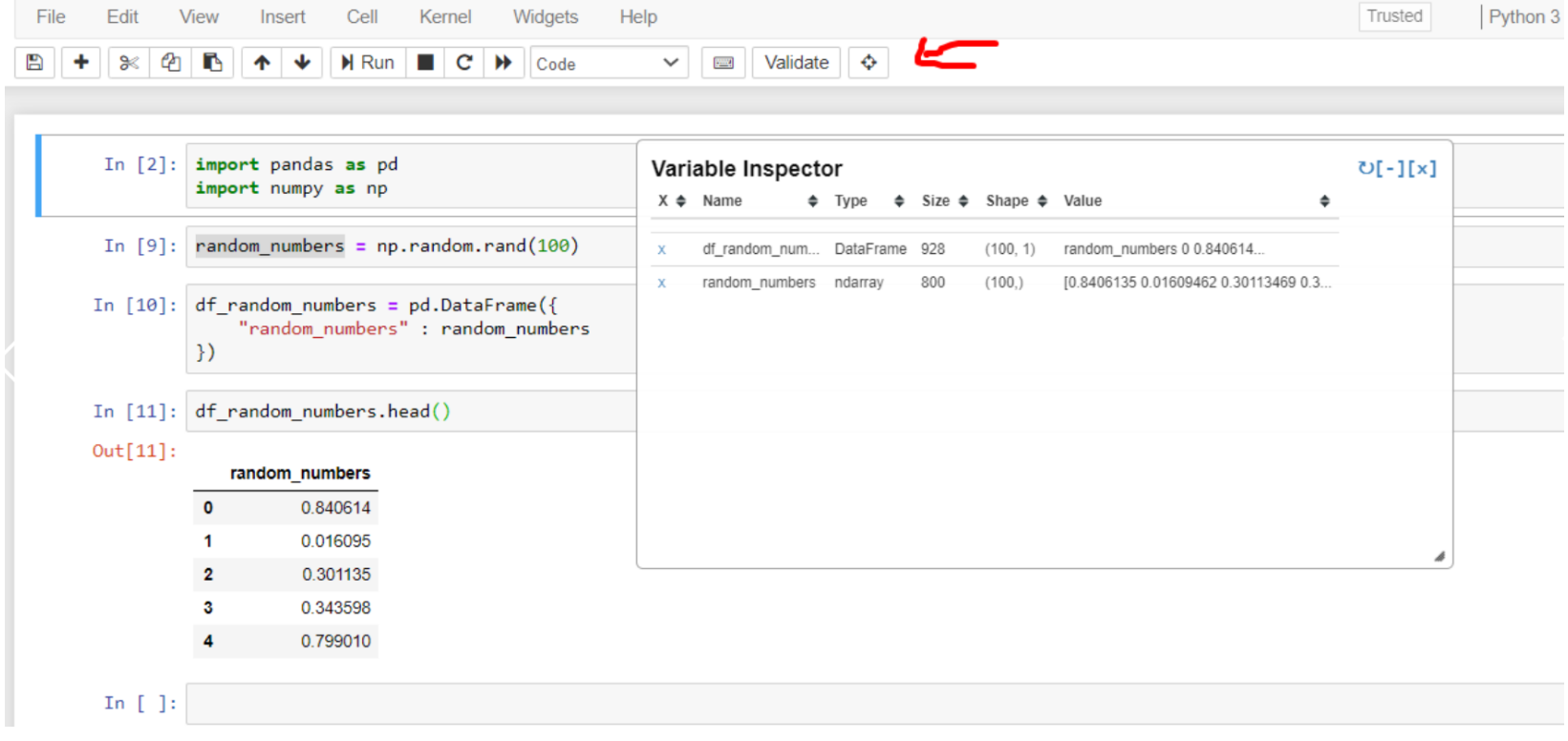
Is it possible to get an extension for jupyter notebook such that you can get an overview of the different variables that you have defined, akin to a "view" in the R environment?

Yes, it is certaintly possible.

The extension that you are looking for is: https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/nbextensions/varInspector/README.html (https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/nbextensions/varInspector/README.html)

A guide to install the nbextension functionality to your jupyter notebook can be found here: https://towardsdatascience.com/jupyter-notebook-extensions-517fa69d2231 (https://towardsdatascience.com/jupyter-notebook-extensions-517fa69d2231),

After you have followed the guide on the previous slide you should be able to do this:



That was pretty much it.

Any other questions?

Note that the format for the course this year is a bit different than last year.

Last year assignment 0 was a bit shorter. Furthermore, it was not interactive, like the one this year.

Thus, it does not come as a surprise that there are not that many questions :)