Computer modeling of physical phenomena

27-28.02.2024

Lab 0: Crash course on Python

Starting

Console

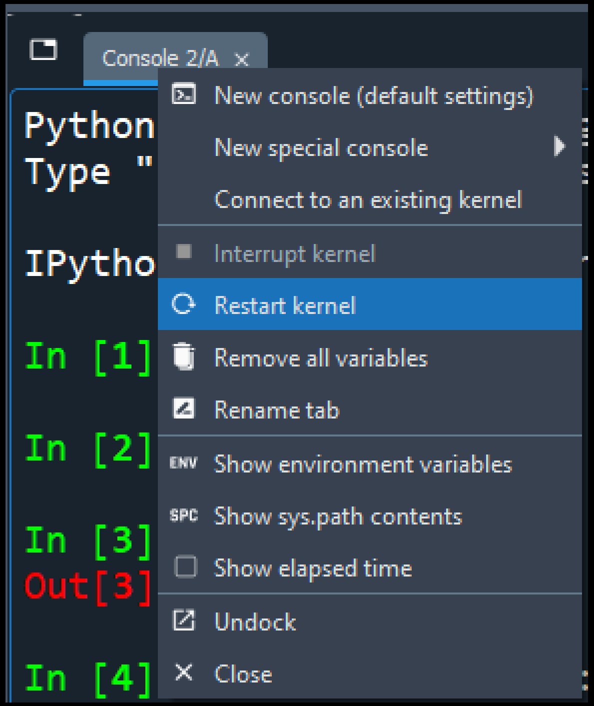
- we enter commands separately (console remembers variables)
- blocks of code (e.g. *for* loop) entered with *shift+enter*
- if we want to clear the variables, we restart the console
- good for a quick check how some function works or what is the value of a variable

```
Console 3/A
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.26.0 -- An enhanced Interactive Python.

In [1]: a = 5

In [2]: b = 3

In [3]: a + b
Out[3]: 8

In [4]: for i in range(a):
   ...:     print (i)
0
1
2
3
4

In [5]:
```

Starting

Console

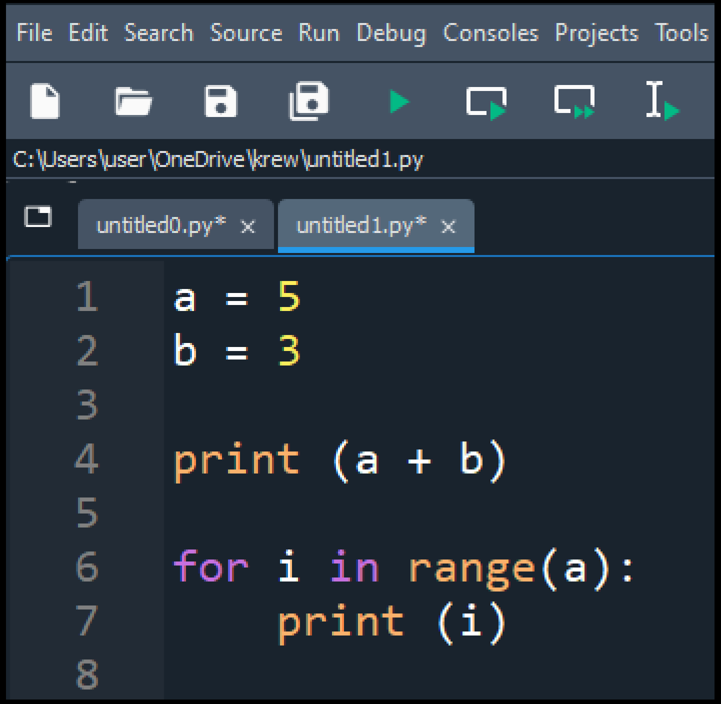
- we enter commands separately (console remembers variables)
- blocks of code (e.g. *for* loop) entered with *shift+enter*
- if we want to clear the variables, we restart the console
- good for a quick check how some function works or what is the value of a variable



Starting

File

- we enter all commands that are to be executed
- with Run (F5) we execute the whole file
- we may see some results in the console
- standard way of coding



Starting

Cells in a file

- we split our code into cells (using *#%%*)
- with *shift+enter* we start a given cell (its result is remembered!)
- after some changes we don't need to rerun the whole file, but only a given cell

```python
#%%
def func(a, b):
    print ("a = ", a, "b = ", b)
    return a + b

m = 12
n = 16

#%%
print(func(m, n))

#%%
```

Basics

Operations

```
a = 2
b = 3
print (a + b)
print (a / b) # floating-point division
print (a // b) # integer division
print (a % b) # remainder from division
print (a ** b) # exponentiation
```

Basics

Variables

```
m = 2.5
n = 4
print (type(m), type(n))
print (m > n) # comparing variables of different types
print (m * n, type(m * n)) # automatic type conversion

m, n = 5, 6. # multiple assignment
t = m, n # t is a tuple
print (t)
print (t[0], t[1], type(t))
```

Basics

Strings

```python
s = "12.3"
print (s * 2) # surprise
print (float(s) * 2)

a = 6.4
print ("a = " + str(a)) # numbers in a string
print (f"a = {a}") # best to use f-string
print (f"a = {2 * a}") # allows operations within strings
print (f"a = {a:.3f}") # easy to set precision
```

Basics

Strings

```
word = "Help" + "A" # combining strings
print (word)
print (word[0:2]) # substrings

word2 = word[:2] + word[3:]
print (word2, len(word2)) # len works also for lists and tuples
```

Basics

List operations

```
# this is a list of strings
b = ["Mary", "had", "a", "little", "lamb"]

print (b[3]) # addressing list elements
print (b[3][2:4]) # ... and letters
print (len(b[3])) # or word lengths
```

Basics

List operations

```
# this is a list of strings
b = ["Mary", "had", "a", "little", "lamb"]

b.append('!') # adding element
print (b)
b.reverse() # reversing order
print (b)
b.pop() # removing element
print (b)
```

Basics

List operations

```
# we can use operators on lists
# (and on almost everything else)

a = [1, 2, 3]
b = [5, 6, 7]

print (a + b)
print (2 * a)
print (a > 1) # but not all operators
```

Basics

for loop

```
# this is a list of strings
b = ["Mary", "had", "a", "little", "lamb"]

for i in range(len(b)): # we iterate on a pointer
    print (i, b[i], len(b[i]))

for x in b: # we iterate on list elements
    print (b.index(x), x, len(x))
```

Basics

while loop

```
# Fibonacci series
# each element is a sum of two previous ones
a, b = 0, 1 # multiple assignment
while b < 10:
    print (b)
    a, b = b, a + b
```

Basics

Functions

```
def fib(n = 3): # definition with a default argument
    a, b = 0, 1 # multiple assignment
    while a < n:
        print (a, end = ' ') # end to print in one line
        a, b = b, a + b


fib(2000)
fib()
print (fib()) # what is returned?
```

Basics

Functions

```
def fib(n = 3): # definition with a default argument
    a, b = 0, 1 # multiple assignment
    while a < n:
        print (a, end = ' ') # end used to print in one line
        a, b = b, a + b
    return a, b


fib(2000)
fib()
print (fib()) # and now?
```

Basics

Local and global variables

```
def f_loc(x):
    print ('x to', x)
    x = 2 # new local variable
    print ('Locally x is', x)

x = 50
f_loc(x)
print ('x is still', x)
```

Basics

Local and global variables

```python
def f_glob():
    global x
    print ('x is', x)
    x = 2
    print ('Globally x changed to', x)

x = 50
f_glob()
print ('x is now', x)
```

Basics

Module ***numpy***

```python
import numpy as np

n = 5
v = np.zeros(n) # numpy array of zeros

for i in range(n):
    v[i] = i / 5

print ('v = ', v)
```

Basics

Module ***numpy***

```
import numpy as np

# more elegant
# range divided with a given step
u = np.arange(0, 1, 0.2)
print ('u = ', u)

# range divided into a fixed number of equally distant steps
w = np.linspace(0, 0.8, 5)
print ('w = ', w)
```

Basics

Module ***numpy***

```
# np.array() is powerful
import numpy as np

a = np.array([1, 2, 3])
b = np.array([5, 6, 7])

print (a + b) # works different than lists
print (2 * a) # probably more intuitive?
print (a > 1) # this time it works

print (np.append(a, b)) # we can append like this
```

Basics

Module ***numpy***

```
# some math
import numpy as np

x = np.exp(1j * np.pi / 4)

print (np.abs(x)) # modulus
print (np.angle(x)) # complex argument
print (np.real(x)) # real part
print (np.imag(x)) # imaginary part
print (np.sqrt(x)) # square root
```

Basics

Plots

```python
# plotting is easy!
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi, np.pi, 100)
y1 = np.cos(x) # mathematical operations in numpy
y2 = np.sin(x) # work for whole arrays

# plot with The Thin Red Line
plt.plot(x, y1, color = 'r', linestyle = '-', linewidth = '2')

# simpler syntax, plot with The Thin Blue Line
plt.plot(x, y2, 'b--', lw = 1)
plt.show() # display the plot
```

Basics

Plots

```
# easy to add labels
# make sure the command are before show

plt.xlabel('x data', fontsize = 20)
plt.ylabel('y results', fontsize = 20)
plt.title('Functions: sine and cosine')

plt.grid(True)
plt.legend(('cos(x)', 'sin(x)'))
```

Basics

Plots

```
# it is also possible to add labels this way
plt.plot(x, np.sin(x), label = 'sin(x)')
plt.plot(x, np.cos(x), label = 'cos(x)')
plt.legend()

# and to save the figure
# (save it before plt.show())
plt.savefig('xy.png')
```

Basics

***Plots* (extra)**

```
# it may be easier to have plots in separate windows
# you can do that with qt5 interface (if it’s not defaultly done in your IDE)
%matplotlib qt5 # it may be necessary to install qt5
fig = plt.figure('Fig1')
fig.clear()
plt.plot(np.sin(x))
plt.plot(np.cos(x))
fig.show()

# to go back to console display
%matplotlib inline
```

Basics

***Plots* (extra)**

```
# you can use Latex in plot labels!
x = np.linspace(-np.pi, np.pi, 100)
plt.plot(x, np.exp(x), label = r'$e^{x}$')
plt.plot(x, 1 / x, label = r'$\frac{1}{x}$')
plt.legend()
plt.xlabel(r'[$-\pi$, $\pi$]')
plt.show()
```

Basics

Task

Make a plot of functions exp(x), sin(x) and x^2 in range [0, 1]. Each function should be plotted with a different colour and line style (continuous, dashed, dotted). Add the legend, labels and plot title. Good luck!


PYTHON
IS STRONG WITH THIS ONE