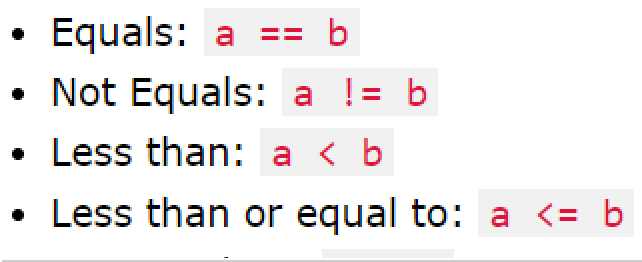
Week 03 – Support notes

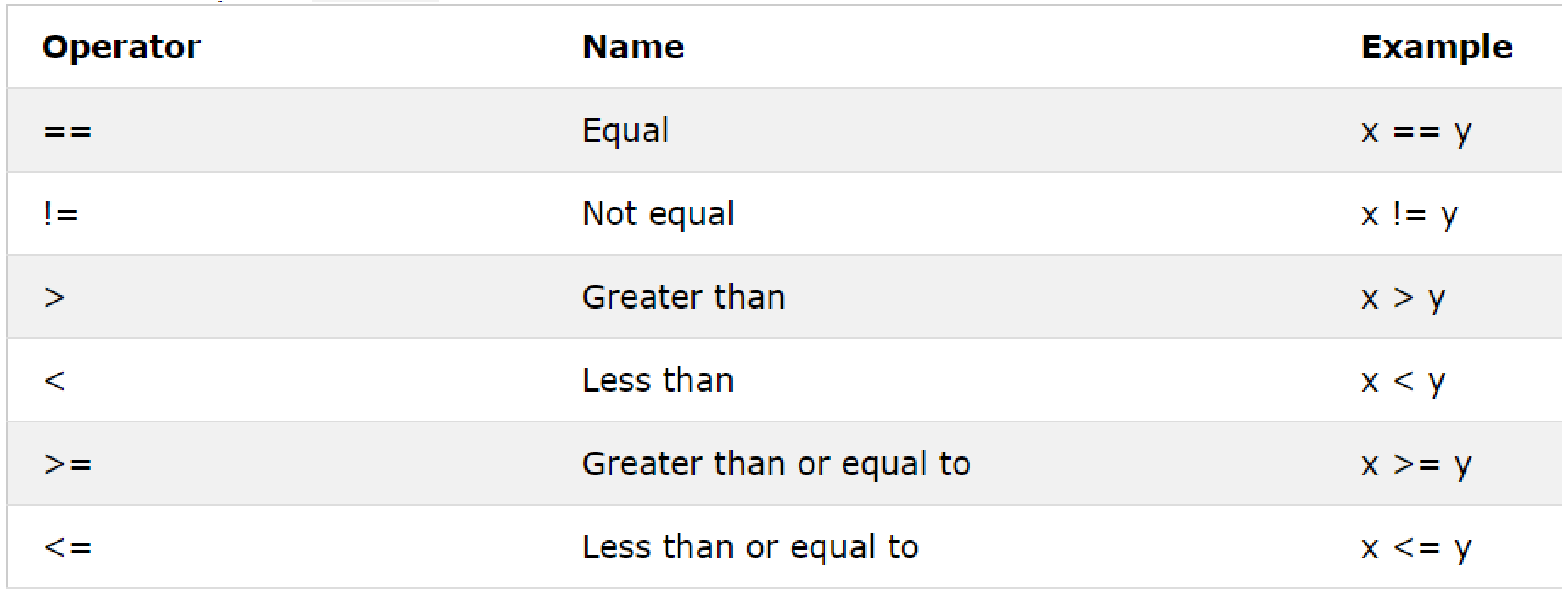
Tópicos

- Expressões lógicas
- Instruções condicionais

Conditions: just to remember



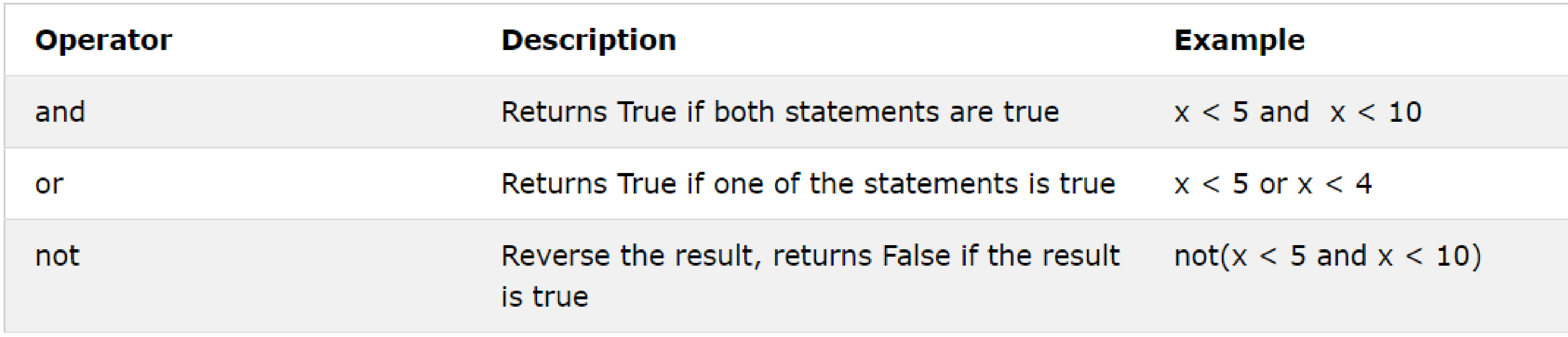
- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

https://www.w3schools.com/python/python_conditions.asp
https://www.w3schools.com/python/python_operators.asp

Boolean: just to remember



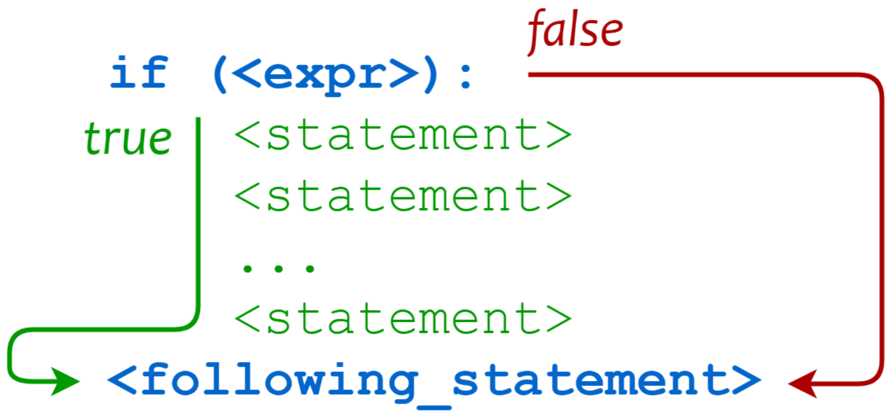
Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

https://www.w3schools.com/python/python_conditions.asp
https://www.w3schools.com/python/python_operators.asp

If and if .. Else

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```



```
if (<expr>):
    <statement>
    <statement>
    ...
    <statement>
<following_statement>
```

https://realpython.com/python-conditional-statements/

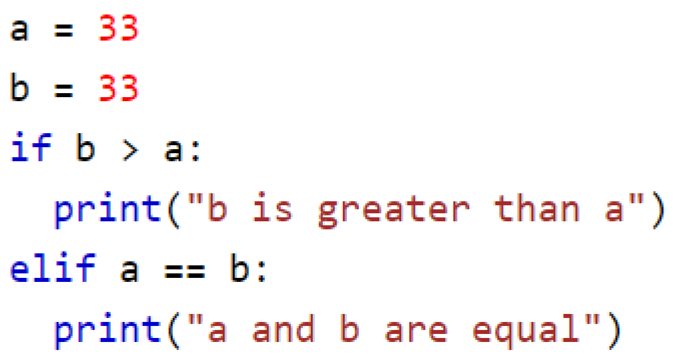
indentation

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

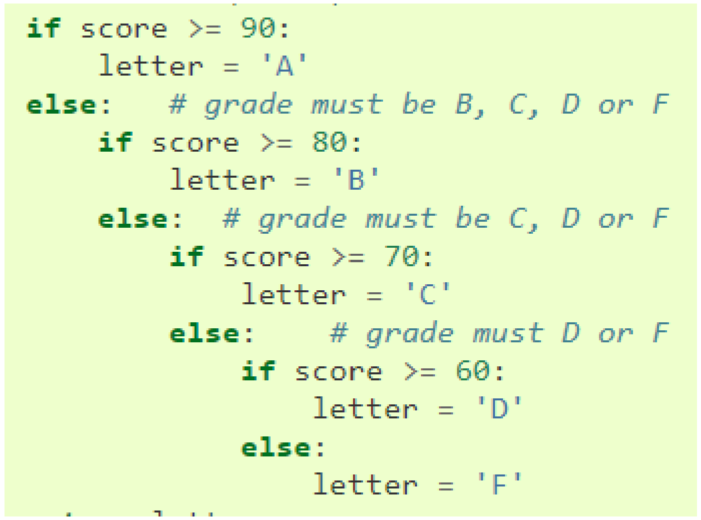
```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

If … elif: Just to remember



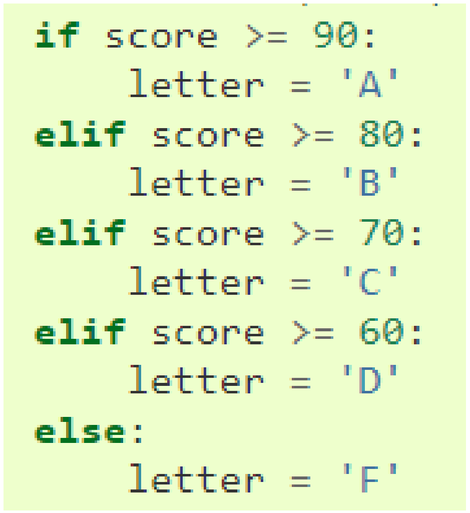
```python
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

If else → if elif

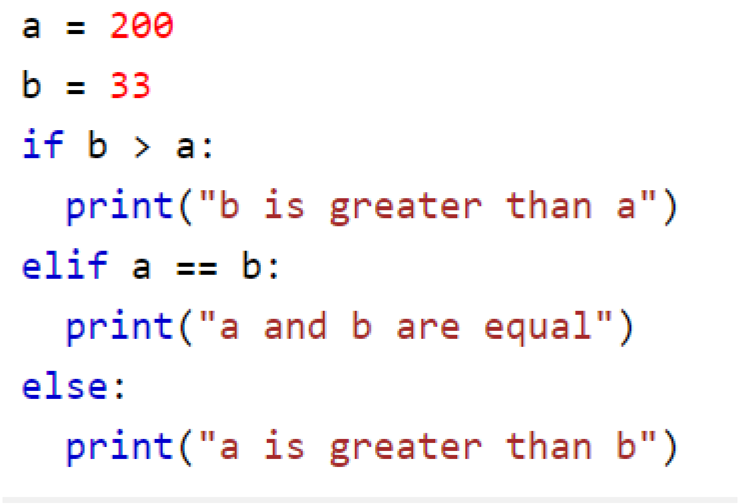


```
if score >= 90:
    letter = 'A'
else:   # grade must be B, C, D or F
    if score >= 80:
        letter = 'B'
    else:  # grade must be C, D or F
        if score >= 70:
            letter = 'C'
        else:     # grade must D or F
            if score >= 60:
                letter = 'D'
            else:
                letter = 'F'
```

<=>



```
if score >= 90:
    letter = 'A'
elif score >= 80:
    letter = 'B'
elif score >= 70:
    letter = 'C'
elif score >= 60:
    letter = 'D'
else:
    letter = 'F'
```



```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

“compressing”

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

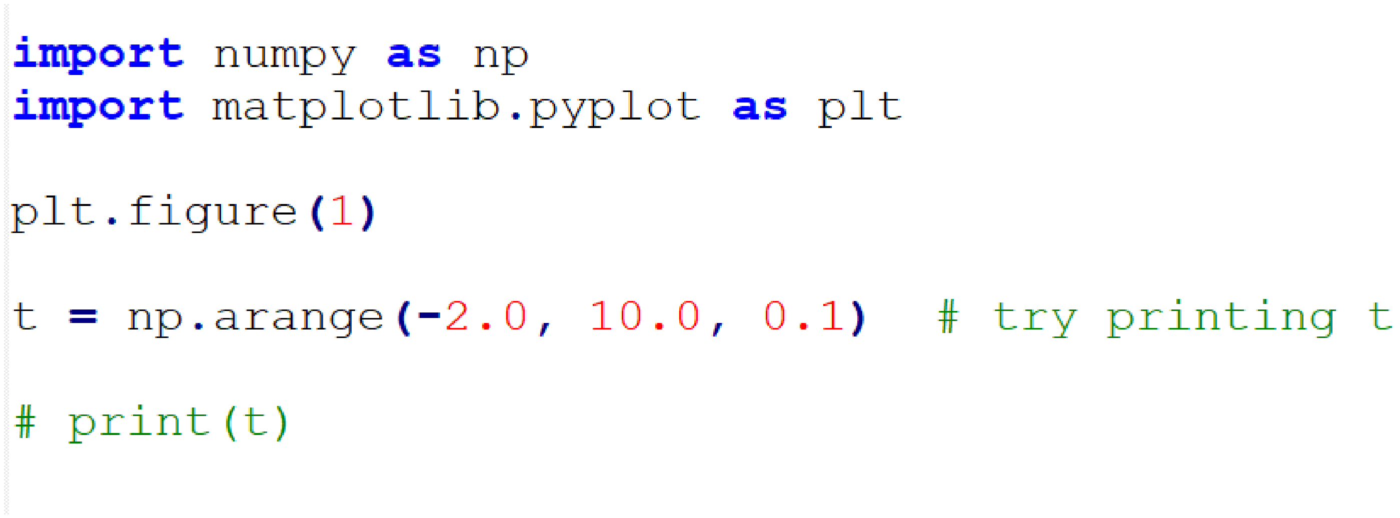
You may find this notation – it is clear and readable

```
a = 2
b = 330
print("A") if a > b else print("B")
```

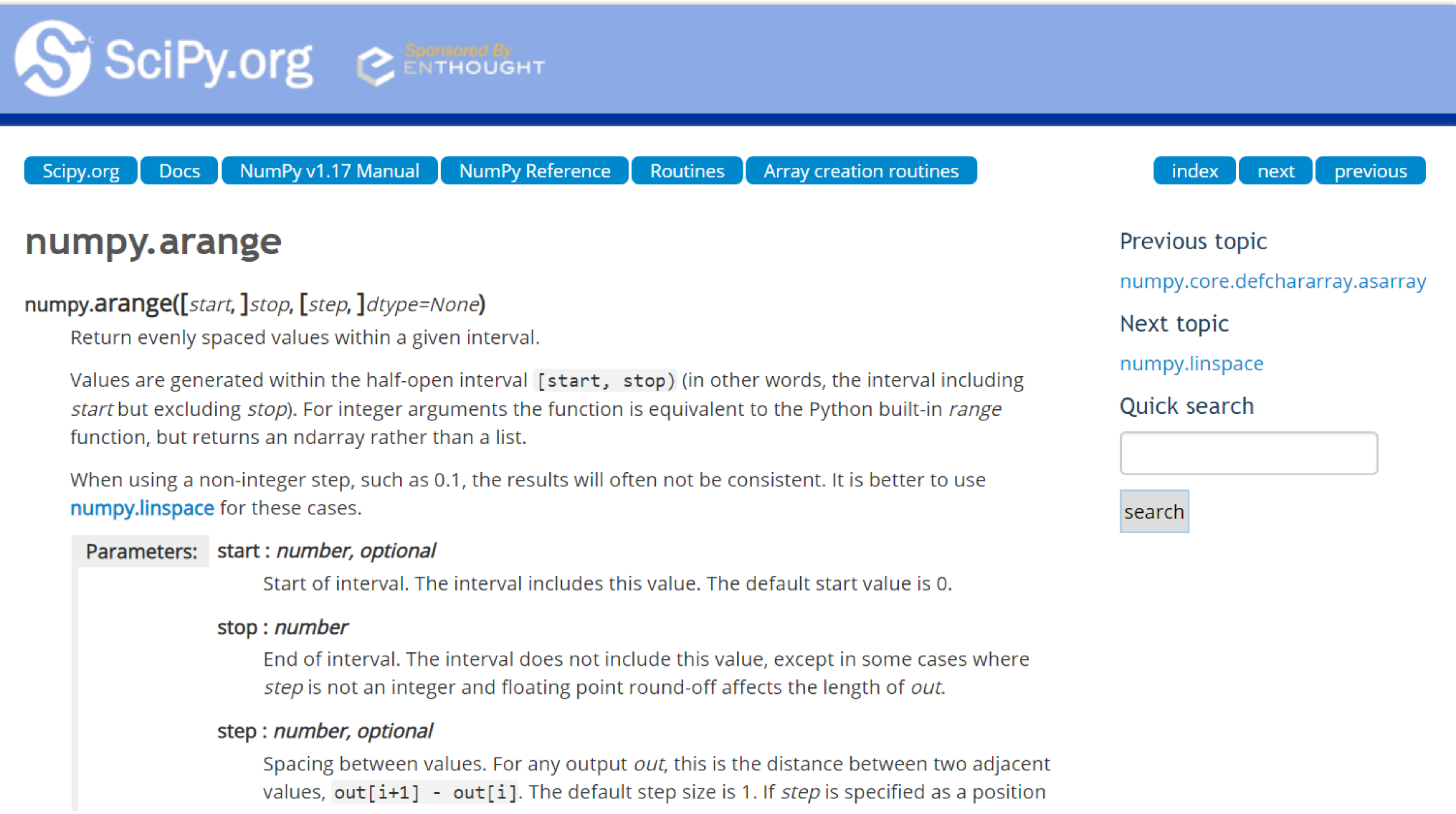
```
if a > b: print("a is greater than b")
```

challenges

- Plot.py (first class)
  - Insert interval to plot function
  - Insert number of divisions
- i.e. Provide the parameters to numpy.arrange



```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(1)

t = np.arange(-2.0, 10.0, 0.1)   # try printing t

# print(t)
```



numpy.arange

numpy.arange([*start*,]*stop*, [*step*,]*dtype=None*)

Return evenly spaced values within a given interval.

Values are generated within the half-open interval `[start, stop)` (in other words, the interval including *start* but excluding *stop*). For integer arguments the function is equivalent to the Python built-in *range* function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use numpy.linspace for these cases.

Parameters: **start** : *number, optional*

Start of interval. The interval includes this value. The default start value is 0.

stop : *number*

End of interval. The interval does not include this value, except in some cases where *step* is not an integer and floating point round-off affects the length of *out*.

step : *number, optional*

Spacing between values. For any output *out*, this is the distance between two adjacent values, `out[i+1] - out[i]`. The default step size is 1. If *step* is specified as a position

Previous topic

numpy.core.defchararray.asarray

Next topic

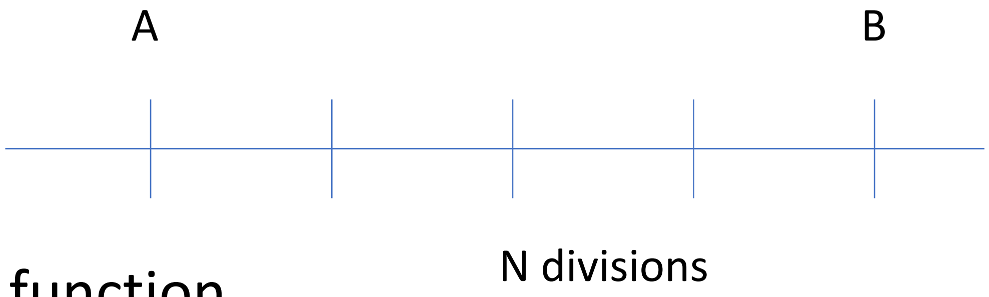
numpy.linspace

Quick search

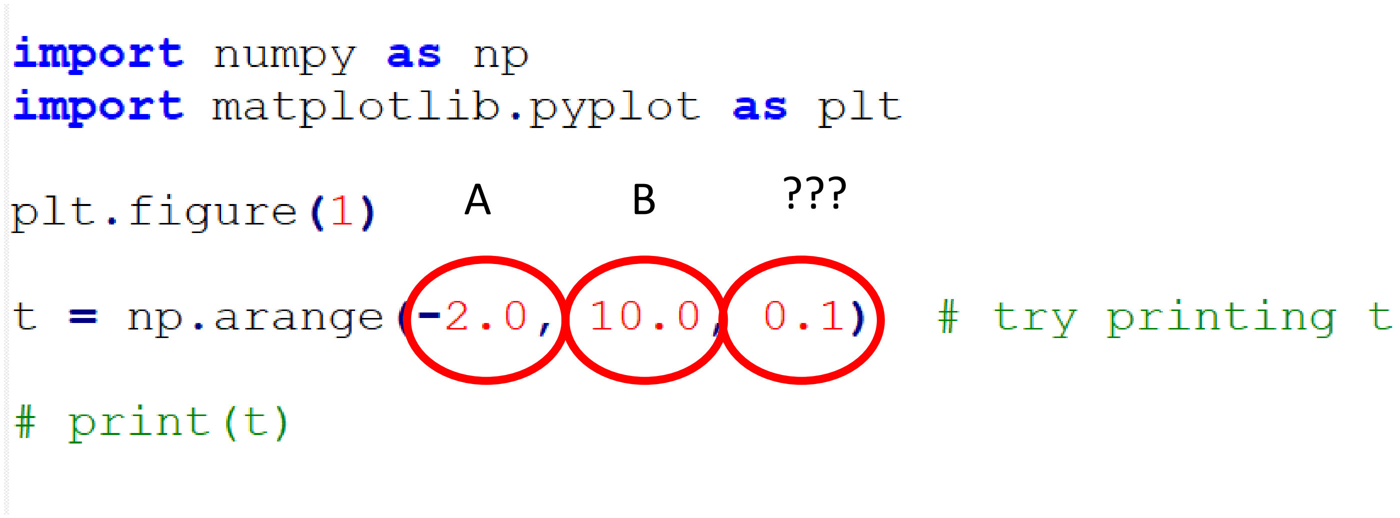
search

https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html

challenges



- Plot
  - Insert interval to plot function
  - Insert number of divisions
- i.e. Provide the parameters to numpy.arrange



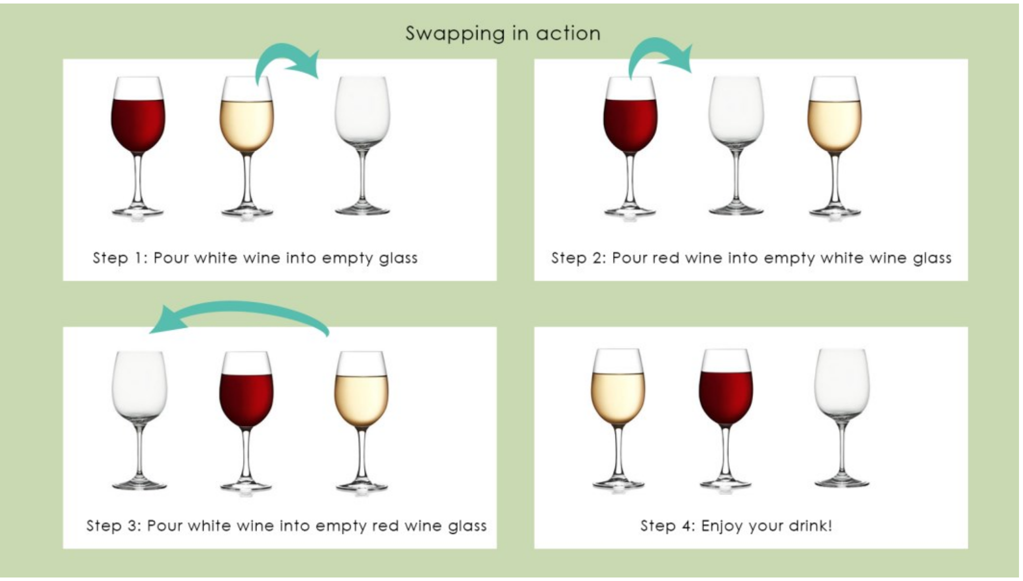
```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(1)

t = np.arange(-2.0, 10.0, 0.1)  # try printing t

# print(t)
```

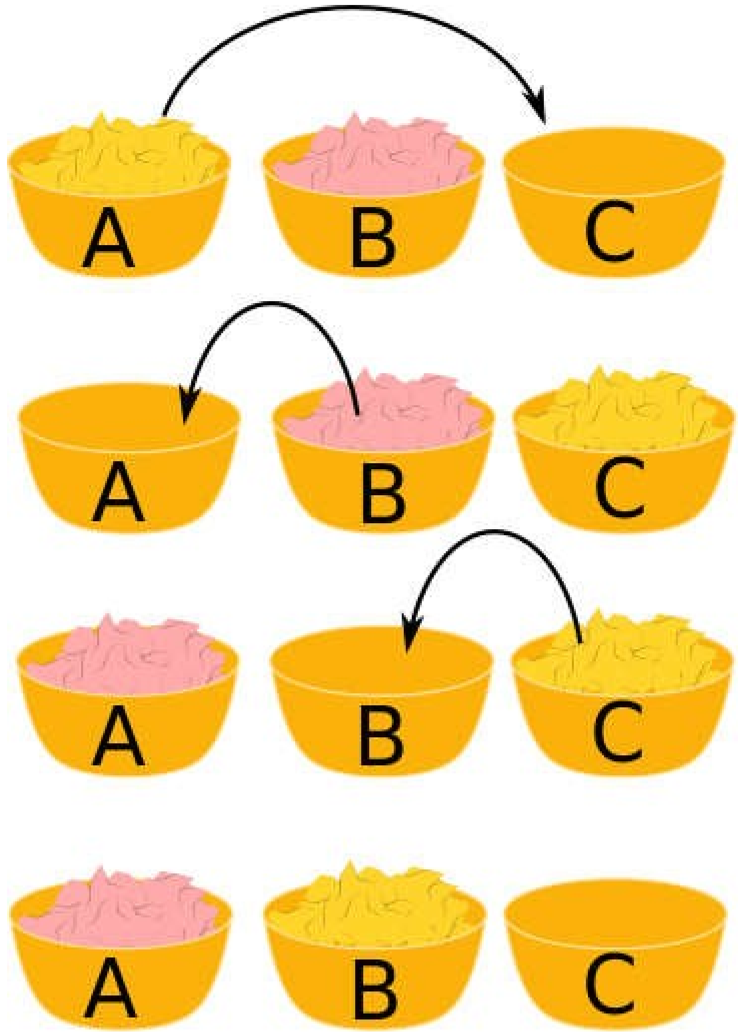
```
#for a valid interval [a,b], a<b
a=float( input("a?"))
b=float( input("b?"))
n= int ( input("n?"))
plt.figure(1)
# need to ensure valid interval
# need to calculate d ( the interval step)
t = np.arange(a, b, d )
```

Problem: inserting an interval

- For a valid interval [a,b]
  - a<b
- BUT if some one inserts as extremes 4 and 2
  - [4,2] is not valid, but most humans will understand it as [2,4]
  - Computers are stupid, they do not do it…
  - You must help them
- Solution
  - Don't do nothing
    - – print that you cannot work with "wrong" interval and end
  - **Try to do something**
    - **swap a,b if a>b and work with a new valid interval**

Swapping: the basic idea


Swapping in action
Step 1: Pour white wine into empty glass
Step 2: Pour red wine into empty white wine glass
Step 3: Pour white wine into empty red wine glass
Step 4: Enjoy your drink!



https://www.i-programmer.info/babbages-bag/498-the-magic-swap.html

swap a,b if a>b

traditional

Python solution

```
temp := a
a := b
b := temp
```

```
a, b = b, a
```

< = >

```
temp = a
a = b
b = c
c = temp
```

```
a, b, c = b, c, a
```

https://www.w3resource.com/python-exercises/python-basic-exercise-91.php
https://www.programiz.com/python-programming/examples/swap-variables
https://www.pythoncentral.io/swapping-values-in-python/

Plot: a draft of the solution

```
#for a valid interval [a,b], a<b
a=float( input("a?"))
b=float( input("b?"))
n= int ( input("n?"))
plt.figure(1)
# need to ensure valid interval
If a>b :
   a,b = b,a
# need to calculate d ( the interval step)
d =  … # finish
t = np.arange(a, b, d )
```

Plot: a draft of the solution

```
#for a valid interval [a,b], a<b
a=float( input("a?"))
b=float( input("b?"))
n= int ( input("n?"))
plt.figure(1)
# need to ensure valid interval
t = np.arange(a, b, d )
If a>b :
   a,b = b,a
# need to calculate d ( the interval step)
d =  … # finish
t = np.arange(a, b, d )
```