MH2401 — Algorithms and Computing III

Handout for weeks 1–2

AY 2018/19, S1

Exercises

Operations

Here we review basic operations and order of precedence in Python. To solve exercises of this section, you can work with the command line Python.

Exercise 1.
Try the following pairs of commands in Python. Why do the pairs of commands produce the same/different outputs?

(a) `-2 ** 4` `(-2) ** 4`

(b) `-2 ** 4 * 3` `-2 ** (4 * 3)`

(c) `2 / 3 * 4` `2 / (3 * 4)`

(d) `2 * 3 / 4` `2 * (3 / 4)`

(e) `3 > 2 > 1` `(3 > 2) > 1`

(f) `2 < 1 and 3 < 4 or 4 < 5` `2 < 1 and (3 < 4 or 4 < 5)`

Solution 1.

(a) The pair produces different outputs because `**` is of higher precedence than unary `-`; the first command outputs the negative of the fourth power of two, while the second outputs the fourth power of negative two.

(b) The pair produces different outputs because `**` is of higher precedence than `*`; the first command outputs the product of `-2 ** 4` and `3`, while the second outputs the negative of the `(4 * 3)`rd power of `2` (i.e., the negative of the `12`th power of `2`).

(c) The pair produces different outputs because `/` and `*` are of the same order of precedence; the first command performs `/` before `*`, while the second performs `*` before `/`.

(d) The pair produces the same output even though `/` and `*` are of the same order of precedence, and the first command performs `*` before `/`, while the second performs `/` before `*`. This is because $\frac{2\times 3}{4}$ and $2 \times \frac{3}{4}$ have the same value.

(e) The pair produces different outputs because the first command compares `3 > 2` and `2 > 1` which are `True`, while the second command compares `3 > 2` first, which results in `True`, then the command compares `True > 1` which is `False`.

(f) The pair produces different outputs because relational operators are of higher precedence than logical operators. Also, here is the precedence order for the boolean operators only (in decreasing order): `==`, `!=`, `and`, `or`. The first command evaluates the truth value of the conjunction of statements `2<1` and `3<4`, followed by comparing with the next statement `4<5` by taking disjunction. On the other hand, in the second command, the disjunction in the bracket is evaluated first before taking the conjunction.

Exercise 2.
Write down a *single* Python command to evaluate each of the following expressions.

(a) $2 \times 4^{-7\times 5/6} - 1$

(b) 2^{2^3}

Solution 2.
(a) `2 * 4 ** (-7 * 5 / 6)-1`

(b) `2 ** (2 ** 3)`

Exercise 3.
Try the following commands with different real values for `a`. When `a` is an arbitrary real number, what outputs do the following Python commands produce? Why?

(a) `0 <= a <= 10`

(b) `a * (a > 0)`

(c) `(a > 5) - (a < 5)`

Solution 3.
(a) This command outputs `True` when `a` is any real number from `0` to `10`; `False` otherwise.

(b) This command outputs `a` when it is positive, and `0` otherwise. This is because the comparison gives `1` (value for `True` when used with operator `*`) when `a` is positive, so that the product is `a`; and the comparison gives `0` (value for `False` when used with operator `*`) otherwise, so that the product is still `0`.

(c) This command outputs `1` when `a > 5`, `0` when `a = 5`, and `-1` when `a < 5`. This is because when `a > 5`, the two comparisons result in `1 - 0`; when `a = 5`, the two comparisons result in `0 - 0`; and when `a < 5`, the two comparisons result in `0 - 1`.

Vector operations

Here we work with vector operations in Python. You will need to import the library *numpy* by typing the command `import numpy as np`. These exercises are harder than exercises in the previous section and it is probably a good idea to use Python scripts here rather than typing commands in the command line.

We will need the following function (it's a good idea to read a manual on it): `arange`. Note that we need to add `np.` to the function's name to specify that it comes from the package *numpy*.

Exercise 4.
Write down a *single* Python command to produce each of the following vectors (loops not allowed) In these vectors, n is a positive integer whose value is given in the Python variable `n`.

(a) $\begin{bmatrix} 2 & 4 & 8 & \cdots & 2^n \end{bmatrix}$

(b) $\begin{bmatrix} 1 & 3 & 6 & \cdots & \frac{n(n+1)}{2} \end{bmatrix}$

Solution 4.
(a) `2 ** np.arange(1,n+1)`

(b) `np.arange(1, n+1) * (np.arange(1, n+1) + 1) / 2`

Explanation: we need to apply a function to each entry of a vector. The function in part (a) is $f(x) = 2^x$ and in part (b) is $f(x) = \frac{x(x+1)}{2}$. The vector in both cases is $(1, 2, \cdots, n)$, which is `np.arange(1, n+1)` in Python. The resulting expression is, in both cases, the result of replacing x in the formula with the `np.arrange(1, n+1)` and writing it in Python notation.

Note that part (b) admits a shorter solution, `np.arange(1, n+1) * np.arange(2, n+2) / 2`.

Matrix generation and matrix operations

Here we will work with matrices in Python. Still, we need to import the library *numpy* by typing the command `import numpy as np`. We will need the following functions (it's a good idea to read a manual on them): `arange`, `matrix`, `append`, `concatenate`, `transpose`. Note that we need to add `np.` to the function's name to specify that it comes from the package *numpy*.

We will review the differences between *array* and *matrix* in *numpy*. Before going on with the next question, run the following commands in Python:

```
a = np.array([[0, 2, -2],
              [1, 1, 3]])

b = np.array([[1, 2, 3],
              [-1, -2, -4]])

A = np.matrix([[0, 2, -2],
               [1, 1, 3]])

B = np.matrix([[1, 2, 3],
               [-1, -2, -4]])
```

Exercise 5.
First, try running `print(a)` and `print(A)`. Note that the output looks the same. Both variables represent the matrix

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix}$$

Try the following pairs of commands in Python. Why do the pairs of commands produce the same/different outputs?

(a) `a + b` `A + B`

(b) `a ** 2` `A ** 2`

(c) `a * b.T` `A * B.T`

(d) `a.dot(b.T)` `a @ b.T`

(e) `np.matrix(a).T * np.matrix(b)` `A.T * B`

Solution 5.

(a) The pair produces the same output. In other case, it is the sum of the two matrices,

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ -1 & -2 & -3 \end{bmatrix} = \begin{bmatrix} 0 & 4 & 4 \\ 0 & -1 & -1 \end{bmatrix}$$

(b) This pair produces different outputs because operations with *array* objects are element-wise operations while operations with *matrix* objects are matrix operations. Therefore, `a ** 2` is

$$\begin{bmatrix} 0^2 & 2^2 & (-2)^2 \\ 1^2 & 1^2 & 3^2 \end{bmatrix}$$

while `A ** 2` is

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix}^2,$$

which is undefined.

(c) This pair produces different outputs because operations with *array* objects are element-wise operations while operations with *matrix* objects are matrix operations. In both cases, `.T` means transposing a matrix and hence the command `a * b.T` is meant co compute the matrix whose entries are products of the corresponding entries of matrices

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & -1 \\ 2 & -2 \\ 3 & -4 \end{bmatrix},$$

which is undefined since the two matrices have different dimensions. On the other hand, the command `A * B.T` calculates the matrix product

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & -1 \\ 2 & -2 \\ 3 & -4 \end{bmatrix},$$

which is a 2×2 matrix.

(d) These commands produce the same output and it's the product

$$\begin{bmatrix} 0 & 2 & -2 \\ 1 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & -1 \\ 2 & -2 \\ 3 & -4 \end{bmatrix},$$

(e) These commands produce the same output because the function `np.matrix` converts its input to a matrix and therefore both commands represent the same matrix operations, the product of a 3×2 matrix and a 2×3 matrix.

Exercise 6.
Write down a *single* Python command to produce each of the following matrices (loops are not allowed). In these matrices, m and n are positive integers whose values are given in the Python variables `m` and `n`, respectively.

(a) $\begin{bmatrix} 0 & 1 & 2 & \cdots & n \\ 1 & 1 & 1 & \cdots & 1 \\ 2 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m & 1 & 1 & \cdots & 1 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & \cdots & j & \cdots & n \\ \vdots & & \vdots & & \vdots \\ i & \cdots & i \times j & \cdots & i \times n \\ \vdots & & \vdots & & \vdots \\ m & \cdots & m \times j & \cdots & m \times n \end{bmatrix}$

Solution 6.
(a) The following command produces a variable of type *array*:

```
np.append([np.arange(n+1)], np.append(np.transpose([np.arange(1,m+1)]), np.ones((m,n)), axis=1), axis=0).
```

The following alternative produces a variable of type *matrix*:

```
np.concatenate((np.matrix(np.arange(n+1)), np.concatenate((np.matrix(np.arange(1,m+1)).T, np.ones((m, n))), axis = 1)), axis = 0)
```

(b)
```
np.matrix(np.arange(1, m+1)).T * np.matrix(np.arange(1, n+1))
```

Note: we can check the type of a Python variable by executing `type()`.

Matrix subsetting

Here we will work with matrices in Python. Still, we need to import the library *numpy* by typing the command `import numpy as np`.

Let's create a 5×6 matrix M in Python by executing the following command:

```
M = np.matrix([[1, -2, 0, -4, 5, 6],
               [7, -8, 0, -10, 11, -12],
               [13, 0, 15, -16, 17, -18],
               [19, -20, 21, -22, 23, -24],
               [25, -26, 0, -28, 29, -30]])
```

Exercise 7.
When `M` is an arbitrary matrix with at least 5 rows and 5 columns, and `i` and `j` are arbitrary integers between 0 and 4, what outputs do the following Python commands produce?

(a) `M[i, j]`

(b) `M[i, :]`

(c) `M[:, j]`

(d) `M[:2, -2:]`

(e) `M[::2, :]`

(f) `M[:, 1::2].shape[0]`

(g) `M[:, 1::2].shape[1]`

(h) `M[-1, :] == 0`

(i) `M[-1, :] = 0`

(j) `np.delete(M, 3, 1)`

Solution 7.

(a) This command outputs the entry in the `(i+1)`th row and `(j+1)`th column of `M`.

(b) This command outputs the `(i+1)`th row of `M`.

(c) This command outputs the `(j+1)`th column of `M`. Note that if `M` were an *array* rather than a *matrix*, then this command would still extract the $(j+1)$th column but it would print it as a row.

(d) This command outputs the submatrix in the first two rows and the last two columns of `M`.

(e) This command outputs the submatrix formed by the odd rows of `M`.

(f) This command outputs the number of rows in the submatrix of `M` formed by all rows and even columns. This is just the same as the number of rows of `M`itself

(g) This command outputs the number of columns in the submatrix of `M` formed by all rows and even columns, i.e., the number of even columns of `M`.

(h) This command outputs a row vector of the same size as a row of `M`, containing only `False` and `True`, with an entry `1=True` when and only when the corresponding entry in the last row of `M` is `0`.

(i) This command sets all the entries in the last row of `M` to be 0.

(j) This command outputs `M` with its fourth column deleted. Note that changing the third argument `1` to `0` will delete the fourth row instead of a fourth column.

Exercise 8.

When `M` is an arbitrary matrix with at least 5 rows and 5 columns, and `i` is an arbitrary integer between 0 and 4, what outputs do the following Python commands produce?

(a) `np.append(M[:2 , 0], M[-2:, -1], axis = 1)`

(b) `M[i]`

(c) `M[np.array([2, 4, 1]), np.array([3, 0, 0])]`

Solution 8.

(a) This command outputs a 2×2 matrix containing the first two entries in the first column and the last two entries in the last column.

(b) This command outputs the `(i+1)`th row of `M`.

(c) This commands outputs the 1×3 matrix whose elements are `M[2, 3]`, `M[4, 0]`, and `M[1, 0]`.

Exercise 9.
Try the following commands with different matrices for `M`. When `M` is an arbitrary matrix, what outputs do the following Python commands produce?

(a) `M > 0`

(b) `np.where(M == 0)`

(c) `M[M < 0]`

(d) `np.any(M > 0)`

(e) `np.all(M >= 0)`

Solution 9.
(a) This command outputs a matrix of the same size as `M` , containing only `True` or `False`, with an entry `True` when and only when the corresponding entry in `M` is positive.

(b) This command outputs an ordered pair `(r,c)`, where `r` records the rows of the zero entries in `M` and `c` records the columns of the corresponding zero entries in `M`, i.e., `(r[i],c[i])` is the position of the `(i+1)`th entry of `M` that is positive.

(c) This command outputs a row vector containing the negative entries of `M`.

(d) This command outputs `True` when and only when there exists a positive entry in `M`, but outputs `False` otherwise.

(e) This command outputs `True` when and only when all entries in `M` are nonnegative, but outputs `False` if there is at least one negative entry.

Exercise 10.
Write a *single* Python command to produce a given matrix `M` with all its negative entries changed to 2401.

Solution 10.
```
M[M < 0] = 2401
```

Extra Questions

Exercise 11.
Write a *single* Python command to output the value of each of the following functions at the real number x, whose value is given in the Python variable `x`.

(a) $f(x) = \begin{cases} x^2 & \text{if } x \geq 1, \\ 1 & \text{if } x < 1. \end{cases}$

(b) $g(x) = \begin{cases} 0.5 & \text{if } \sin(x) > 0.5, \\ \sin(x) & \text{if } -0.5 \leq \sin(x) \leq 0.5, \\ -0.5 & \text{if } \sin(x) < -0.5. \end{cases}$

Solution 11.

(a) `x ** 2 * (x >= 1) + (x < 1)`

(b) `0.5 * (sin(x) > 0.5) + sin(x) * (-0.5 <= sin(x) <= 0.5) - 0.5 * (sin(x) < -0.5)`

For the last question, we need to create a *square* matrix X of size $n \times n$. Below is a Python command that creates the 9×9 matrix X whose $(i.j)$th entry is $\cos i \times \sin j$ rounded to 2 decimal places:

```
n = 9
X = np.round(np.matrix(np.cos(np.arange(1, n+1))).T *
             np.matrix(np.sin(np.arange(1, n+1))) , 2)
```

Exercise 12.

Use linear indexing to perform each of the following exercises with a *single* Python command. Note that "linear indexing" means that you are not supposed to use *numpy* functions, such as `diag`:

(a) Extract the main diagonal of a square matrix `X`. In other words, form the vector `X[0, 0], X[1, 1], ..., X[n-1, n-1]`.

(b) Extract the anti-diagonal of a square matrix `X`, i.e., form the vector `X[n-1, 0], X[n-2,1], ..., X[0, n-1]`.

(c) Flip the matrix `X` about the anti-diagonal, i.e., form a new matrix whose (i, j) th entry is the same as the $(n - j, n - i)$ th entry of `X`.

Solution 12.

(a) `X[range(n),range(n)]`

(b) `X[range(n-1,-1,-1),range(n)]`

(c) `X[::-1 , ::-1].T`

Exercise 13.

Create each of the following matrices A of size $n \times n$ using a single Python command (no loops are allowed):

(a) $a_{ij} = \min(i, j)$, i.e.,

$$A = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2 & \cdots & 2 \\ 1 & 2 & 3 & \cdots & 3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 3 & \cdots & n \end{bmatrix}$$

(b) The matrix whose entries above the anti-diagonal are 0 and on the anti-diagonal and below it 1. For instance, for $n = 5$, we would have

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Solution 13.

(a)
```
np.minimum(np.matrix(np.arange(1, n+1)).T * np.ones((1, n)) , np.ones((1, n)).
    T * np.matrix(np.arange(1, n+1)))
```

(b)
```
0 + (np.matrix(np.arange(1, n+1)).T * np.ones((1, n)) - np.ones((1, n)).T * np
    .matrix(np.arange(n, 0, -1)) >= 0)
```

Please submit your feedback on this lab. For instance, the course instructors need to know if this handout is sufficient to understand the material and if lab assistants are helpful and their explanations are clear.

- `https://tinyurl.com/mh2401-week1-feedback`