1 Coding cheat sheet BIOS1100 H17

Limited to chapters 1-7 of the book. Also to be used during the exam.

1.1 Variables

Syntax	Description
`int()`	Converts the argument to integer
`float()`	Converts the argument to float
`round()`	Rounds a number to a certain decimal point

1.2 Lists

Syntax	Description	Result
`L = []`	Initialize an empty list	`[]`
`L = [1, 4.4, "bacteria"]`	Initialize a list	`[1, 4.4, "bacteria"]`
`len(L)`	number of elements in list `L`	`3`
`L.append(2)`	Add 2 to the end of `L`	`[1, 4.4, "bacteria", 2]`
`L.insert(1, "a")`	Insert `"a"` before index `1`	`[1, "a", 4.4, "bacteria"]`
`L[1]`	Index a list, get element `1`	`4.4`
`L[-1]`	Get last element in a list	`"bacteria"`
`L[1:3]`	Slice: copy data to sublist	`[4.4, "bacteria"]`
`del L[1]`	Delete an element (index 2)	`[1, "bacteria"]`
`L.index(4.4)`	Find index of first occurrence of `4.4`	`1`
`L + [1, 3]`	Merge two lists	`[1, 4.4, "bacteria", 1, 3]`
`L.count("bacteria")`	Count occurrences of `"bacteria"`	`1`
`L.copy()`	Copy the list	`[1, 4.4, "bacteria"]`

Results below shown on the list `L = [4, 2, 10]`:

Syntax	Description	Result
`min(L)`	The smallest element in `L`	`2`
`max(L)`	The largest element in `L`	`10`
`sum(L)`	Add all elements in `L`	`16`
`sorted(L)`	Return sorted version of list `L`	`[2, 4, 10]`

1.3 `range`

Syntax	Description
`range(stop)`	From 0 up to, but not including, `stop` with step size 1
`range(start, stop)`	From `start` up to, but not including, `stop` with step size 1
`range(start, stop, step)`	From `start` up to, but not including, `stop` with step size `step`

1.4 Arrays

Syntax	Description
`array([5, 6, 7, 8])`	Convert a list to an array
`zeros(N)`	With `N` zeros
`arange(stop)`	From 0 up to, but not including, `stop` with step size 1
`arange(start, stop)`	From `start` up to, but not including, `stop` with step size 1
`arange(start, stop, step)`	From `start` up to, but not including, `stop` with step size `step`

Some array operations when we have two arrays of equal length, `a = array([1, 2, 3])` and `b = array([1, 2, 3])`:

Syntax	Description	Result
`len(a)`	Number of elements in array `a`	`3`
`a[1]`	Index the array, get element at index one	`2`
`a[1:3]`	Slice: get a view of the data	`array([2, 3])`
`a.copy()`	Creates a copy of an array	`array([1, 2, 3])`
`a + b`	Element-wise addition	`array([2, 4, 6])`
`a + 2`	Add `2` to each element of `a`	`array([3, 4, 5])`
`a - b`	Element-wise subtraction	`array([0, 0, 0])`
`a - 2`	Subtract `2` from each element of `a`	`array([-1, 0, 1])`
`a*b`	Element-wise multiplication	`array([1, 4, 9])`
`a*2`	Multiply each element of `a` with `2`	`array([2, 4, 6])`
`a/b`	Element-wise division	`array([1, 1, 1])`
`a/2`	Divide each element of `a` with `2`	`array([0.5, 1., 1.5])`
`a**b`	Element-wise power	`array([1, 4, 27])`
`a**2`	Each element of `a` to the power of `2`	`array([1, 4, 9])`
`sqrt(a)`	The square root of each element in `a`	`array([1., 1.41421356, 1.73205081])`

1.5 Dictionary

A Dictionary is an unordered collection of object where each value in the dictionary is associated with a key, called a key-value pair. An example of a dictionary is:

```
D = {"A": 0, "G": 2, 100: 2}
```

Strings, floats, integers and several other object not encountered yet can be used as keys. In the table below some important dictionary operations are shown, always using the dictionary `D = {"A":0, 100:2}`.

Syntax	Description	Result
`D = { }`	Initialize an empty dictionary `D`	`{}`
`D = {"A":0, 100:2}`	Initialize a dictionary `D`	`{"A":0, 100:2}`
`D["C"] = 10`	Set or create a key `"C"` with value 10	`{"A":0, 100:2, "C":10}`
`D["A"]`	Value associated with key `"A"`	`1`
`D.get("A")`	Value of `"A"` if `"A"` is in `D`, else `None`	`1`
`"A" in D`	Check if `"A"` is in `D`	`True`
`len(D)`	Number of key value pairs in `D`	`3`
`del D["A"]`	Remove `"A"` and its value from `D`	`{"A":0, 100:2}`
`D.keys()`	Get a view of all keys in `D`	`dict_keys(["A", 100])`
`D.values()`	Get a view of all values in `D`	`dict_values([0, 2])`
`D.copy()`	Copy a dictionary `D`	`{"A": 0, 100: 2}`

Looping over all elements in a dictionary:

```
for key in my_dict:
    print("The key is", key, "and value is", my_dict[key])
```

1.6 Loops

For-loops. A `for` loop repeats a set of statements a specific number of times. It tells the computer that for each element in a sequence (array, list, and others) it should “do something“

```
numbers = [1, 2, 3]

for number in numbers:
    print(number)

print("Finished printing numbers to screen!")
```

While loops. A `while` loop repeats a set of statements as long as a specific condition is met:

```
a = 0
while a < 5:
    # do something with a
    a = a + 1
```

Using enumerate to get the index in a for-loop. The `enumerate()` function gives access to the index and the element for each item in a list.

```
l = ["A", "B", "C", "D"]

for index, element in enumerate(l):
    print("index:", index, ", element:", element)
```

1.7 Functions

A function is given input through *arguments* and gives output using a *return* statement:

```python
def add(a, b):
    """
    Returns the sum of the inputs.
    """
    return a + b

print(add(2, 3))
```

Default function values. A function can have *default values* that are given along with the arguments:

```python
def add(x, y=0, z=0):
    print("x =", x, ", y =", y, ", z =", z)
    return x+y+z

print(add(1, 3))
print(add(1, z=2))
```

Global and local variables. Variables defined inside a function are not available outside the function:

```python
def my_function():
    inside = 1  # local variable
    return inside

outside = my_function()
print(inside)
```

1.8 If tests

If-else tests

```python
color = "red"

if color == "red":
    print("The color is red!")
else:
    print("The color is not red!")
```

Using elif

```python
codon = "UAG"

if codon == "UAA":
    print("codon is a stop codon")
elif codon == "UGA":
    print("codon is a stop codon")
elif codon == "UAG":
    print("codon is a stop codon")
else:
    print("codon is not a stop codon")
```

Logical operators for combining boolean expressions. Boolean values (True and False) represent truth values of logic.

The keywords and and or combine multiple truth statements in the same if test.

```
my_number = 4

if my_number > 2 and my_number < 5:
    print("my_number is between 2 and 5!")
else:
    print("my_number is not between 2 and 5!")
```

The or-keyword allows you to test if any of the two expressions are True.

```
my_number = 6

if my_number < 2 or my_number > 5:
    print("my_number is smaller than 2, or larger than 5!")
else:
    print("my_number is something else!")
```

Comparison operators. Comparison operators compare expressions on both sides of the operator and return True or False.

Code	Meaning
a == b	a is equal to b
a != b	a is not equal to b
a < b	a is less than b
a > b	a is greater than b
a <= b	a is less than or equal to b
a >= b	a is greater than or equal to b
a in b	a is an element in the list b

The keyword not can be inserted in front of a boolean expression to change the value from True to False, or from False to True.

1.9 Random choice in Python

The choice() function that picks one element at random from a list:

```
from pylab import *
parent_1 = ['B', 'b']
allele_1 = choice(parent_1)
```

1.10 Short-hand syntax for common operations

Code	Equivalent code
`n += 1`	`n = n + 1`
`n -= 1`	`n = n - 1`
`n *= 1`	`n = n*1`
`n /= 1`	`n = n/1`

1.11 Reading and writing from files

Reading using pandas

```python
import pandas
data = pandas.read_csv("ecoli.csv")
# convert data in columns to lists
t = list(data["t"])
E = list(data["E"])
```

1.12 Plotting

Syntax	Description
`xlabel("Time, t (minutes)")`	Label for x-axis
`ylabel("Population size, E")`	Label for y-axis
`title("Measured bacterial population growth")`	Title of figure
`plot(t, E, "g-", label="Population at 39 C")`	Plot `t` and `E` as a green line `"g-"` with a label
`plot(t2, E2, "yo", label="Population at 29 C")`	Plot `t2` and `E2` as yellow circles `"yo"` with a label
`legend()`	Show the legend in the plot
`subplot(2, 1, 1)`	plot in 2 rows, 1 columns, first (top left) plot
`yscale("log")`	Use logarithmic axis on the y-axis
`savefig("name_of_plot.png")`	Save the plot as `name_of_plot.png`
`show()`	Show the plot