W9: Intro to Python - Day 2

Fei-Man Hsu, PhD

UCLA QCBio Collaboratory, Fall 2025

This is the notebook for the second day of the Collaboratory Workshop, *Intro to Python*.

Outline

Day 1:

- Concept and setup
- Basic data types and variables
- If statements

Day 2:

- Loops
- More data types: Collections (lists, sets, dictionaries)
- Reading/Writing Files

Day 3:

- Functions
- Libraries

Assignments are mandatory for credits

- Due by the end of Oct 17

Recap

- Basic data types: String, Integer, Float, Boolean
- Variables and literals
- Reading input from a user and writing output to display
- Boolean operators and logical expressions
- If statement

Variables and Literals

```
my_variable = "My first variable!"

   (variable)     (literal)
```

Basic data types: `str`, `int`, `float`, and `bool`

```
v1 = "Hi there!"
v2 = 300
v3 = 300.0
v4 = 10 < 14
```

Reading input from a user and writing output to display

```
i1 = int(input("Input any integer: "))
print("{} + {} = {}".format(i1, i1, i1 + i1))
```

Relational operations and boolean operators

```
not ((5 > 4) or (5 <= 4) and (5 <= 4))
```

```
In [1]: not ((5 > 4) or (5 <= 4) and (5 <= 4))
```

```
Out[1]: False
```

If statement

```
a = 2
if a > 5:
    print(1)
elif a == 5:
    print(2)
else:
    print("Here we go!")
```

```
In [2]: a = 2
        if a > 5:
            print(1)
        elif a == 5:
            print(2)
        else:
            print("Here we go!")
```

```
Here we go!
```

Warm up

Write a program that checks if the year from a user's input is a leap year.

Hint: To check if a year is a leap year, divide the year by 4. If it is fully divisible by 4, it is a leap year. For example, the year 2016 is divisible 4, so it is a leap year, whereas, 2015 is not. However, century years like 300, 700, 1900, 2000 need to be divided by 400 to check whether they are leap years or not. While century years not divisible by 400 is not a leap year, century years divisible by 400 is a leap year.

```
In [4]: year = int(input("Input year: "))
        if year % 4 == 0: # % is the remainder
            if (year % 100 == 0): # century year?
                if year % 400 == 0:
                    print("Yes, a leap year")
                else:
                    print("No, not a leap year")
            else:
                print("Yes")
        else:
            print("No")
```

```
Input year: a
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[4], line 1
----> 1 year = int(input("Input year: "))
      2 if year % 4 == 0: # % is the remainder
      3     if (year % 100 == 0): # century year?

ValueError: invalid literal for int() with base 10: 'a'
```

Write a program for a biotech company that synthesize primers (oligonucleotides composed of 4 letters A, T, C and G).

1. Ask the customer to provide the sequence
2. Check the length is equal or greater than 20nt and less than 26nt
3. The CG content is equal or greater than 40% and less than 60% If any condition doesn't match, print "Not ideal". If passes all criteria, print "Ideal".

```
In [5]: user_primer = input("Oligonucleotide: ") #string datatype
        countA = user_primer.count('A') # count is a method to count occurence of a substring
        countC = user_primer.count('C')
        countG = user_primer.count('G')
        countT = user_primer.count('T')
        length = countA + countC + countG + countT
        GCcontent = (countC + countG)/length
        if (20 <= length < 26) and (0.4 <= GCcontent < 0.6):
            print('Ideal')
```

```
else:
    print('Not ideal')
```

```
Oligonucleotide: TACGGAAATTACGATTACCCG
Ideal
```

Repeating code in loops -- For loop

Task 1.

For a website, it is required to write a program that validates the password:

- password should be between 12 to 16 characters in length;
- password should have at least 2 uppercase characters;
- password should have at least 4 lowercase characters;
- password should have at least 1 number;

- Input: A string from user input
- Output: Output a single string, either Valid if the password is valid, or Invalid otherwise

Collection

- **Collection** is a group of objects with a logical connection.
- String works as a collection of characters.
- for loops are convenient for working with each item in collections.

In [9]:
```
# for character in "Hello!":
#     print("\n") # TAB
#     print(character, end = ',') # print every char in "Hello!" with ',' delimiter
print(help(print))
```

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.

None
```

What is the sum of first 10 nonnegative integers?

```
s = 0
for i in range(10):
    s += i
print(s)
```

```
45
```

In [12]:
```
# s = 0
# for i in range(10):
#     print(i)
#     s = s + i # s += i
# print(s)
print(range(10))
help(range)
```

```
range(0, 10)
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(...)
 |      Return a reverse iterator.
 |
 |  count(...)
 |      rangeobject.count(value) -> integer -- return number of occurrences of value
 |
 |  index(...)
 |      rangeobject.index(value) -> integer -- return index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

- The variable `s` is called accumulator (remember this technique for future)
- The `range(n)` function creates a collection of nonnegative integers
  - It begins with `0`, and it ends with `n - 1`.
- `range()` may have other arguments that can be found in the documentation

Documentation

Documentation keeps rules, hints, and examples how to use built in functions.

How to find documentation:

- the `help()` function
- Google
- Stack Overflow
- Official documentation at https://docs.python.org/
- Use search engines including web page `find` tool.

In [13]:
```
#help(range)
for i in range(0, 10, 3):
    print(i, end = ',')
```

```
0,3,6,9,
```

How to define a range with step size of 2 with the `range()` function and get a sequence of 1, 3, 5, 7, 9?

Run `help("")` and find `step` on the page.

In [15]:
```
for i in range(1, 10, 2):
    print(i, end=',')
```

```
1,3,5,7,9,
```

In [16]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

Function documentation also shows up when you press `Shift-Tab` with the function name on Jupyter Notebook.

How to check if a symbol or character is uppercase, lowercase, or digit?

- Find the correct method
  - A method is an action that belongs to a specific object of certain type.

In [ ]:
```
help(str)
```

In [17]:
```
'E'.isupper()
```

Out[17]:
```
True
```

In [19]:
```
"Ep".islower()
```

```
Out[19]: False
```

```
In [20]: "3".isdigit()
```

```
Out[20]: True
```

Now, back to Task 1.

For a website, it is required to write a program that validates the password:

- password should be between 12 to 16 characters in length;
- password should have at least 2 uppercase characters;
- password should have at least 4 lowercase characters;
- password should have at least 1 number;

- Input: A string from user input
- Output: Output a single string, either Valid if the password is valid, or Invalid otherwise

```
In [21]: password = input("Input password: ") # USER INPUT
chars = 0
uppers = 0
lowers = 0
digits = 0
for c in password:
    chars += 1
    if c.isupper():
        uppers += 1 # uppers = uppers + 1
    elif c.islower():
        lowers += 1
    elif c.isdigit():
        digits += 1

if (12 <= chars <= 16) and (uppers >= 2) and (lowers >= 4) and (digits >= 1):
    print("Valid")
else:
    print("Invalid")
```

```
Input password: BioInfo2026qcBIO
Valid
```

Task 2.

For a website, it is required to write a program that validates a username:

- username should have 8 characters;
- the first character in a username should be an upper letter;
- the last character in a username should be a number;

- Input: A string from user input

- Output: Print a single string, either `Valid` if the username is valid, or `Invalid` otherwise

Index access for string characters

We can access characters in strings using indices in brackets.

```
In [22]: "Hello"[0] # index starts with 0.
```

```
Out[22]: 'H'
```

```
In [23]: "Hello"[1]
```

```
Out[23]: 'e'
```

```
In [24]: "Hello"[-1] # for the last character.
```

```
Out[24]: 'o'
```

```
In [25]: "Hello"[-2] # second-to-last.
```

```
Out[25]: 'l'
```

Also we can do slicing to obtain a substring:

```
In [31]: "Hello"[-2:] # 0, 1, 2
```

```
Out[31]: 'lo'
```

This begins with the index 1, and ends with index -2, without including the index -2.

```
In [29]: "Hello"[1:4]
```

```
Out[29]: 'ell'
```

```
In [32]: s = "Hello"
         s[1:4]
```

```
Out[32]: 'ell'
```

`len()`

The `len()` function returns the number of elements in a collection.

```
In [33]: len(s)
```

```
Out[33]: 5
```

Back to Task 2.

For a website, it is required to write a program that validates a username:

- username should have 8 characters;
- the first character in a username should be a capitalized letter;
- the last character in a username should be a number;

- Input: A string from user input

- Output: Print a single string, either `Valid` if the username is valid, or `Invalid` otherwise

```
In [35]: username = input("Input username: ")
         if (len(username) == 8) and (username[0].isupper()) and (username[-1].isdigit()):
             print("Valid")
         else:
             print("Invalid")
```

```
Input username: Bio Inf8
Valid
```

Task 3.

Read n from user and print a triangle following the pattern:

```
1
12
123
1234
12345
...
12345...n
```

For example, if `n = 3`, the output should be:

```
1
12
123
```

`range()` revisited

The `range()` function returns a sequence of numbers, starting from `0` by default, and increments by `1` (by default), and stops before a specified number.

```
for i in range(10):
    print(i)
```

In [36]:
```
for i in range(10):
    print(i, end = ',')
```

```
0,1,2,3,4,5,6,7,8,9,
```

`stop` and `step` in `range()`

Print first 10 non-negative integers in a reverse order.

`help(range)` said:

```
|  range(stop) -> range object
|  range(start, stop[, step]) -> range object
|
|  Return an object that produces a sequence of integers from start (inclusive)
|  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
|  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
|  These are exactly the valid indices for a list of 4 elements.
|  When step is given, it specifies the increment (or decrement).
```

```
range(9, -1, -1)
```

In [37]:
```
for i in range(9, -1, -1):
    print(i, end = ',')
# help(print)
```

```
9,8,7,6,5,4,3,2,1,0,
```

Combining `range()` and `len()`

Accessing characters of the string by index in for loop.

```
x = "Hello!"
for i in range(len(x)):
    print(x[i])
```

In [40]:
```
x = "Hello!"
for i in range(len(x)):
    print(i, x[i])
# for i in x:
#     print(i)
```

```
0 H
1 e
2 l
3 l
4 o
5 !
```

Another option: `enumerate()`

```
x = "Hello!"
for (i, c) in enumerate(x):
    print("Character in position ", i, ": ", c)
```

In [41]:
```
help(enumerate)
```

```
Help on class enumerate in module builtins:

class enumerate(object)
 |  enumerate(iterable, start=0)
 |
 |  Return an enumerate object.
 |
 |    iterable
 |      an object supporting iteration
 |
 |  The enumerate object yields pairs containing a count (from start, which
 |  defaults to zero) and a value yielded by the iterable argument.
 |
 |  enumerate is useful for obtaining an indexed list:
 |      (0, seq[0]), (1, seq[1]), (2, seq[2]), ...
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
```

In [42]:
```
x = "Hello!"
for (i, c) in enumerate(x):
    print("Character in position ", i, ": ", c)
```

```
Character in position  0 :  H
Character in position  1 :  e
Character in position  2 :  l
Character in position  3 :  l
Character in position  4 :  o
Character in position  5 :  !
```

printing on the same line

- `print()` has a parameter called `end`. It sets the symbol that should be sent to output after printing the string.
- by default `end='\n'`

```
for c in "Hello":
  print(c, end="\t")
```

In [46]:
```
for c in "Hello":
    print(c, end = " ")
```
```
H e l l o
```

Exercise (5 min)

1) Print word string "Hello!" in reverse order. **Hint**: In range(start,stop[,step]), start is inclusive, stop is exclusive.

In [48]:
```
s = "Hello!"
for i in range(len(s)-1, -1, -1):
#     print(i, s[i])
    print(s[i], end = '')
```
```
!olleH
```

In [49]:
```
s = "Hello!"
for i in range(-1, -len(s)-1, -1):
    #print(i, s[i])
    print(s[i], end="")
```
```
!olleH
```

In [1]:
```
#s[0:3]
s[::-1] # The first colon represents the end of the index
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 #s[0:3]
----> 2 s[::-1]

NameError: name 's' is not defined
```

2) Triple all 'o' in a user's phrase. For example: "Hello world! Yo!" should be printed as

"Hello-o-o wo-o-orld! Yo-o-o!"

Hints:

- Use `input()` to read user's phrase
- Use parameter `end=""` in `print()`

In [2]:
```
s = input("Input string: ")
for c in s:
    if c == 'o':
        print("o-o-o", end="")
    else:
        print(c, end="")
```
```
Input string: Hello World, Yo!
Hello-o-o Wo-o-orld, Yo-o-o!
```

Nested loops

- for loop can be nested.
- printing a multiplication table:

  ```
  for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end='\t')
    print()
  ```

In [3]:
```
for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end="\t")
    print() # Next line when j0 to j10 are iterated
```

```
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100
```

Exercise.

Print multiplication table with the following format:

```
1	|	2	3	4	5	6	7	8	9	10
--------------------------------------------------------------------------------
2	|	4	6	8	10	12	14	16	18	20
3	|	6	9	12	15	18	21	24	27	30
4	|	8	12	16	20	24	28	32	36	40
5	|	10	15	20	25	30	35	40	45	50
6	|	12	18	24	30	36	42	48	54	60
7	|	14	21	28	35	42	49	56	63	70
8	|	16	24	32	40	48	56	64	72	80
9	|	18	27	36	45	54	63	72	81	90
10	|	20	30	40	50	60	70	80	90	100
```

In [4]:
```
for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end="\t")
        if j == 1:
            print("|", end="\t")
    print()
    if i == 1:
        print("-" * 80)
```

```
1	|	2	3	4	5	6	7	8	9	10
--------------------------------------------------------------------------------
2	|	4	6	8	10	12	14	16	18	20
3	|	6	9	12	15	18	21	24	27	30
4	|	8	12	16	20	24	28	32	36	40
5	|	10	15	20	25	30	35	40	45	50
6	|	12	18	24	30	36	42	48	54	60
7	|	14	21	28	35	42	49	56	63	70
8	|	16	24	32	40	48	56	64	72	80
9	|	18	27	36	45	54	63	72	81	90
10	|	20	30	40	50	60	70	80	90	100
```

Back to Task 3.

Read n from user and print a triangle following the pattern:

```
1
12
123
1234
12345
...
12345...n
```

For example, if `n = 3`, the output should be:

```
1
12
123
```

In [5]:
```
n = int(input("Input n: "))
for i in range(1, n + 1):
    for j in range(1, i + 1):
        print(j, end="")
    print() # Next line
```

```
Input n: 3
1
12
123
```

Indefinite loops

Task 4.

Count the total number of digits in an integer number.

Hint: cast integer to string and count

In [6]:
```
i1 = 45234
len(str(i1))  # "45234"
```

Out[6]:
```
5
```

How would we do it without converting it to a string?

While loop

While loops repeat the code until the condition is no longer satisfied.

```
j = 10
while j > 5:
    print(j)
    j = j - 1 # or equivalently,  j -= 1
```

In [8]:
```
j = 10
while j > 5: # Keep looping when j > 5
    print(j)
    j = j - 1  # j -= 1
```

```
10
9
8
7
6
```

In conditions or conversion to `bool`, empty string (`""`), `0`, and `0.0` are interpreted as `False`. Nonempty string and any nonzero numeric values are interpreted as `True`.

In [9]:
```
bool("")
```

Out[9]:
```
False
```

In [10]:
```
bool(0)
```

Out[10]:
```
False
```

In [11]:
```
bool(0.0)
```

Out[11]:
```
False
```

In [12]:
```
bool("t")
```

Out[12]:
```
True
```

In [13]:
```
bool(1)
```

Out[13]:
```
True
```

In [14]:
```
bool(1.1)
```

Out[14]:
```
True
```

Back to Task 4.

Count the total number of digits in an input positive integer number using a while loop.

Hint: Decimal numerical system

In [16]:
```python
i = int(input("Input an integer: "))
count = 0 # digit of the i
while i > 0: # 100 -> 10 -> 1
    i //= 10 # i = i / 10
    count += 1
print(count)
len(str(45234))
```

```
Input an integer: 45234
5
```
Out[16]:
```
5
```

Task 5.

Write a chatbot such that:

- it repeats user's phrases except for questions (sentence ending with a question mark)
- answers "I don't know." when user asks questions
- stops conversation when user says nothing

`break` and `continue`

- the `break` command serves for premature exit from a loop
- the `continue` command serves for skipping one cycle

```python
n = 5
for i in range(1000):
    if i > n:
        break
    print(i)
```

In [17]:
```python
n = 5
for i in range(1000): # 0,1,2 ...999
    if i > n:
        break
    print(i)
```

```
0
1
2
3
4
5
```

```python
s = "Hello, world!"
for c in s:
    if c == "e" or c == "o":
        continue
    print(c, end="")
```

In [18]:
```python
s = "Hello, world!"
for c in s:
    if c == "e" or c == "o":
        continue
    print(c, end="")
```

```
Hll, wrld!
```

Now, back to the task 5.

Write a chatbot such that:

- it repeats user's phrases except for questions (sentence ending with a question mark)
- answers "I don't know." when user asks questions
- stops conversation when user says nothing

In [19]:
```
#bool('')
while True:
    s = input("Say something: ")
    if not s: # "AB" is True, "" is False
        break
    if s[-1] == "?":
        print("I don't know")
#   elif s[-1] == "!":
#     print("Wow")
    else:
        print(s)
```

```
Say something: How are you?
I don't know
Say something: HELLO!
HELLO!
Say something:
```

Exercise (5 min)

Complete the following code to print the numbers in a range using `continue` and `break`

```
start = int(input())
stop = int(input())
for i in range(1000):
    <your code>
```

Ask user to set the range using variables `start` and `stop`.

If `start` is 10 and `stop` is 15, the desired output is:

```
10
11
12
13
14
```

In [20]:
```
start = int(input("Start:" ))
stop = int(input("End:"))
for i in range(1000):
    if i < start:
        continue
    if i >= stop:
        break
    print(i)
```

```
Start:10
End:25
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

Lists

Task 6.

Print an integer in a binary format

- Input: ask user to enter an integer
- Output: print integer in binary format

Binary representation of an integer is the sequence of remainders of successive division of an integer by 2, reversed.

Algorithm:

- in a while loop:
    - divide a number by two
    - store quotient and reminder
    - stop when number is zero
- print the results

List

List is a one of the tools to collect multiple data. For example, list is capable to store average high temperature in Westwood

```
In [21]: temp_Westwood_avg = [41, 44, 50, 58, 67, 76, 85, 84, 77, 64, 49, 40]
```

List can keep one's favorite fruits:

```
In [22]: my_favorite_fruits = ["bananas", "mango", "apples"]
```

List can collect different data types, e.g., it can contain both temperatures and fruits in the same list:

```
In [23]: temp_Westwood_avg + my_favorite_fruits
Out[23]: [41, 44, 50, 58, 67, 76, 85, 84, 77, 64, 49, 40, 'bananas', 'mango', 'apples']
```

`+` concatenates two lists, as it does in strings.

Many things common with strings:

Lists can be multiplied by integer

```
In [24]: [1, 2, 3] * 3 # multiply the elements in the list
Out[24]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

`len()` defines the length of a list

```
In [25]: len([1, 2, 3] * 3)
Out[25]: 9
```

Lists can be iterated like strings:

```
In [26]: for el in [1, 2, 3] * 3:
             print(el, end=",")

1,2,3,1,2,3,1,2,3,
```

Lists can be tested for existence of an element in the list with the keyword `in`:

```
In [27]: 2 in [1, 2, 3]
Out[27]: True
```

```
In [28]: 4 not in [1, 2, 3]
Out[28]: True
```

Elements of the list can be accessed and sliced:

```
In [30]: numbers = ["one", "two", "three", "four", "five"]
```

```
In [31]: numbers[2:4] # index2, 3
Out[31]: ['three', 'four']
```

Slicing can also be nested: try

```
numbers[2][2]
```

In [32]:
```
numbers[2][2]
```
Out[32]:
```
'r'
```

In [33]:
```
numbers[1:3]
```
Out[33]:
```
['two', 'three']
```

```
numbers[-2][:-1]
```

In [34]:
```
numbers[-2][:-1] # "four"
```
Out[34]:
```
'fou'
```

List can even keep another list as an internal element that can be also accessed by inner index

In [36]:
```
bar = [1, "one", ["two", 2.0], [0, 1, 2, 3]]
```

Try:

```
bar[-1][2]
```

In [37]:
```
bar[-1][2]
```
Out[37]:
```
2
```

and:

```
bar[2][0][1]
```

In [38]:
```
bar[2][0][1]
```
Out[38]:
```
'w'
```

`.append()` method

New elements can be added at the end of the list using list's method `.append()`.

```
a = [1, 2, 3]
a.append(0)
a # to show the elements
```

In [39]:
```
a = [1, 2, 3]
print(a)
a.append(0)
print(a)
```
```
[1, 2, 3]
[1, 2, 3, 0]
```

List vs string

Then, what is different between string and list? Strings are *immutable*, while lists are *mutable*. You can assign values to an element of a list.

In [40]:
```
s = "Hello!"
```

In [41]:
```
s[-2] = "0"
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[41], line 1
----> 1 s[-2] = "0"

TypeError: 'str' object does not support item assignment
```

One may modify the string going through a list:

```
ls = list(s)
ls[-2] = "0"
ls
```

In [42]:
```
print(s)
ls = list(s)
print(ls)
ls[-2] = "0"
ls
```

```
Hello!
['H', 'e', 'l', 'l', 'o', '!']
```
Out[42]:
```
['H', 'e', 'l', 'l', '0', '!']
```

In [43]:
```
str(ls)
```
Out[43]:
```
"['H', 'e', 'l', 'l', '0', '!']"
```

Let's make it back to a string using `.join()` method:

```
s = "".join(ls)
s
```

In [44]:
```
"".join(ls)
```
Out[44]:
```
'Hell0!'
```

Method documentations

To find out what methods a certain class has, there is a function called `dir()`. We can find out more about certain function using the `help()` method

```
dir("")
```

In [45]:
```
alist = [1, '1']
print(dir(alist))
# print(alist.count('1'))
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__do
c__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash
__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__
mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setat
tr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'exte
nd', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

```
help("".join)
```

In [46]:
```
help("".join)
```

```
Help on built-in function join:

join(iterable, /) method of builtins.str instance
    Concatenate any number of strings.

    The string whose method is called is inserted in between each given string.
    The result is returned as a new string.

    Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
```

Exercise (3 min)

- Find out what methods lists have. **Hint:** use `[]` or `list` as an argument of `dir()`
- Read about `sort` and `reverse` functions of list.

And try:

```
l = [2, 1, 3]
l.sort()
print(l)
```

```
l.reverse()
print(l)
```

In [51]:
```
l = [2, 1, 3]
# dir(l)
# help(.sort)
# l = [2, 1, 3]
l.sort()
print(l)
l2 = sorted(l)
print(l2)
l.reverse()
print(l)
```

```
[1, 2, 3]
[1, 2, 3]
[3, 2, 1]
```

Now, Back to Task6

Print an integer in a binary format

- Input: ask user to enter an integer
- Output: print integer in binary format

Binary representation of an integer is the sequence of remainders of successive division of an integer by 2, reversed.

Algorithm:

- in a while loop:
  - divide a number by two
  - store quotient and reminder
  - stop when number is zero
- print the results

In [52]:
```
i = int(input("Input integer: "))
l = [] # empty list as acuumulator
while i > 0:
    l.append(i % 2) # remainder
    i //= 2 # i = i / 2

l.reverse()
l_str = []
for el in l:
    l_str.append(str(el))
print("".join(l_str))
```

```
Input integer: 1024
10000000000
```

Reading and writing files

- You can open files on your file system for reading and writing.
- Python can open many file formats including archives by using appropriate libraries.
- Here we are only talking about plain text files.
- Reading and writing from files is similar to interacting with users via keyboard and display
- Reading and writing files allows automatic parsing, analysing, and creating reports in text format.
- For reading and writing there is function `open()`.

Writing a file

```
with open("my_first_file.txt", "w") as f_out:
    f_out.write("Hello, my first file!")
    f_out.write("I am learning Python.")
    f_out.write("Have a nice day!")
```

In [ ]:
```
with open("my_first_file.txt", "w") as f_out:
    f_out.write("Hello, my first file!")
    f_out.write("I am learning Python.")
    f_out.write("Have a nice day!")
```

It is sometimes an issue if a file is written in a program and is not closed. The `with` block avoids this program by deleting/closing `f_out` when the block ends.

In [ ]:
```
f_out = open("my_first_file.txt", "w")
f_out.write("Hello, my first file!")
f_out.write("I am learning Python.")
f_out.write("Have a nice day!")
f_out.close() # Must close the file
```

Note the the file is overwritten since we create and write to the same file name.

Reading a File

```
with open("my_first_file.txt") as f_in:
    for line in f_in:
        print(line)
```

In [ ]:
```
with open("my_first_file.txt") as f_in:
    for line in f_in:
        print(line)
```

Something is off here. We want multiple lines. The writer **does not put a new line** automatically when writing to a file.

Writing a file - second attempt

```
with open("my_first_file.txt", "w") as f_out:
    f_out.write("Hello, my first file!")
    f_out.write("\n")
    f_out.write("I am learning Python.\n")
    f_out.write("Have a nice day!")
    f_out.write("\n")
```

In [ ]:
```
with open("my_first_file.txt", "w") as f_out:
    f_out.write("Hello, my first file!")
    f_out.write("\n")
    f_out.write("I am learning Python.\n")
    f_out.write("Have a nice day!")
    f_out.write("\n")
```

Then, let's try reading it again:

In [ ]:
```
with open("my_first_file.txt") as f_in:
    for line in f_in:
        print(line)
```

Too many lines this time.

As discussed yesterday, the method `.strip()` allows removing white spaces from borders of the line. The code is writing in an additional `\n` at the end of each line it read in.

In [ ]:
```
with open("my_first_file.txt") as f_in:
    for line in f_in:
        print(line.strip())
```

or, you could just use `end=""` for the print function.

In [ ]:
```
with open("my_first_file.txt") as f_in:
    for line in f_in:
        print(line, end="")
```

Assignment 4

Write a program to sum all the running integers from 1 and 1000, that are divisible by 3, 5 or 7, but NOT by 15, 21, 35 or 105.

```
In [ ]:
```

Assignment 5

Write a program to print all the leap years between AD999 and AD2010. Also print the total number of leap years.

Hints:

1. Use a int variable called count, which is initialized to zero
2. Increment the count whenever a leap year is found
3. To check if a year is a leap year, divide the year by 4. If it is fully divisible by 4, it is a leap year. For example, the year 2016 is divisible 4, so it is a leap year, whereas, 2015 is not. However, Century years like 300, 700, 1900, 2000 need to be divided by 400 to check whether they are leap years or not. Century years not divisible by 400 are not leap years.

```
In [ ]:
```

Assignment 6

Use a loop to compute the following series:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots\right)$$

Geometric series:

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots$$

Harmonic series:

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots$$

Alternating series:

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots$$

```
In [ ]:
```