Зертханалық жұмыс - №2

Тізімдер, кортеждер және жолдар

Жұмыстың мақсаты: Python программалау тілімен тәжірибеде танысу

In [ ]:
```
l = [1, 2, 45, 56, "Spell", 2.5, [1,2,3,4], {'name':'Daulet'}, ()]
t = (1,2,4, "Spell", 2.5, (3,4,5), [3,4,5])
s = "Spam"
```

In [ ]:
```
l[0]
```

Out[ ]:
```
1
```

In [ ]:
```
t[0]
```

Out[ ]:
```
1
```

In [ ]:
```
s[0]
```

Out[ ]:
```
'S'
```

In [ ]:
```
len(l)
```

Out[ ]:
```
9
```

In [ ]:
```
len(t)
```

Out[ ]:
```
7
```

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

Out[ ]:
```
4
```

In [ ]:
```
l[0:3]
```

Out[ ]:
```
[1, 2, 45]
```

In [ ]:
```
a = l[0:4]
```

In [ ]:
```
a
```

Out[ ]:
```
[1, 2, 45, 56]
```

In [ ]:
```
a1 = l[0::2]
```

In [ ]:
```
a1
```

Out[ ]:
```
[1, 45, 'Spell', [1, 2, 3, 4], ()]
```

In [ ]:
```
l
```

Out[ ]:
```
[1, 2, 45, 56, 'Spell', 2.5, [1, 2, 3, 4], {'name': 'Daulet'}, ()]
```

In [ ]:
```
l + [4,4,4,4]
```

Out[ ]:
```
[1, 2, 45, 56, 'Spell', 2.5, [1, 2, 3, 4], {'name': 'Daulet'}, (), 4, 4, 4, 4]
```

In [ ]:
```
s
```

Out[ ]:
```
'Spam'
```

In [ ]:
```
"Dangerous" + s
```

Out[ ]:
```
'DangerousSpam'
```

In [ ]:
```
t
```

Out[ ]:
```
(1, 2, 4, 'Spell', 2.5, (3, 4, 5), [3, 4, 5])
```

In [ ]:
```
t  + (3,4,5)
```

Out[ ]:
```
(1, 2, 4, 'Spell', 2.5, (3, 4, 5), [3, 4, 5], 3, 4, 5)
```

In [ ]:
```
s
```

Out[ ]:
```
'Spam'
```

In [ ]:
```
s*3
```

Out[ ]:
```
'SpamSpamSpam'
```

In [ ]:
```
t
```

Out[ ]:
```
(1, 2, 4, 'Spell', 2.5, (3, 4, 5), [3, 4, 5])
```

In [ ]:
```
t * 2
```

Out[ ]:
```
(1,
 2,
 4,
 'Spell',
```

```
 2.5,
 (3, 4, 5),
 [3, 4, 5],
 1,
 2,
 4,
 'Spell',
 2.5,
 (3, 4, 5),
 [3, 4, 5])
```

In [ ]:
```
l*3
```

Out[ ]:
```
[1,
 2,
 45,
 56,
 'Spell',
 2.5,
 [1, 2, 3, 4],
 {'name': 'Daulet'},
 (),
 1,
 2,
 45,
 56,
 'Spell',
 2.5,
 [1, 2, 3, 4],
 {'name': 'Daulet'},
 (),
 1,
 2,
 45,
 56,
 'Spell',
 2.5,
 [1, 2, 3, 4],
 {'name': 'Daulet'},
 ()]
```

In [ ]:
```
s
```

Out[ ]:
```
'Spam'
```

In [ ]:
```
s.upper()
```

Out[ ]:
```
'SPAM'
```

In [ ]:
```
s.isupper()
```

Out[ ]:
```
False
```

In [ ]:
```
s.islower()
```

Out[ ]:
```
False
```

In [ ]:
```
l
```

Out[ ]:
```
[1, 2, 45, 56, 'Spell', 2.5, [1, 2, 3, 4], {'name': 'Daulet'}, ()]
```

In [ ]:
```
l.append(16)
```

In [ ]:
```
l
```

Out[ ]:
```
[1, 2, 45, 56, 'Spell', 2.5, [1, 2, 3, 4], {'name': 'Daulet'}, (), 16]
```

In [ ]:
```
s
```

Out[ ]:
```
'Spam'
```

In [ ]:
```
s1 = "Hello Python! I love you!"
```

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

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(self, format_spec, /)
 |      Return a formatted version of the string as described by format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(...)
 |
 |  __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.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __sizeof__(self, /)
 |      Return the size of the string in memory, in bytes.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case and the rest low
er
 |      case.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for caseless comparisons.
 |
 |  center(self, width, fillchar=' ', /)
 |      Return a centered string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  count(...)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring sub in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
 |
 |  encode(self, /, encoding='utf-8', errors='strict')
 |      Encode the string using the codec registered for encoding.
 |
```

```
|      encoding
|        The encoding in which to encode the string.
|      errors
|        The error handling scheme to use for encoding errors.
|        The default is 'strict' meaning that encoding errors raise a
|        UnicodeEncodeError.  Other possible values are 'ignore', 'replace' and
|        'xmlcharrefreplace' as well as any other name registered with
|        codecs.register_error that can handle UnicodeEncodeErrors.
|
|  endswith(...)
|      S.endswith(suffix[, start[, end]]) -> bool
|
|      Return True if S ends with the specified suffix, False otherwise.
|      With optional start, test S beginning at that position.
|      With optional end, stop comparing S at that position.
|      suffix can also be a tuple of strings to try.
|
|  expandtabs(self, /, tabsize=8)
|      Return a copy where all tab characters are expanded using spaces.
|
|      If tabsize is not given, a tab size of 8 characters is assumed.
|
|  find(...)
|      S.find(sub[, start[, end]]) -> int
|
|      Return the lowest index in S where substring sub is found,
|      such that sub is contained within S[start:end].  Optional
|      arguments start and end are interpreted as in slice notation.
|
|      Return -1 on failure.
|
|  format(...)
|      S.format(*args, **kwargs) -> str
|
|      Return a formatted version of S, using substitutions from args and kwargs.
|      The substitutions are identified by braces ('{' and '}').
|
|  format_map(...)
|      S.format_map(mapping) -> str
|
|      Return a formatted version of S, using substitutions from mapping.
|      The substitutions are identified by braces ('{' and '}').
|
|  index(...)
|      S.index(sub[, start[, end]]) -> int
|
|      Return the lowest index in S where substring sub is found,
|      such that sub is contained within S[start:end].  Optional
|      arguments start and end are interpreted as in slice notation.
|
|      Raises ValueError when the substring is not found.
|
|  isalnum(self, /)
|      Return True if the string is an alpha-numeric string, False otherwise.
|
|      A string is alpha-numeric if all characters in the string are alpha-numeric
and
|      there is at least one character in the string.
|
|  isalpha(self, /)
|      Return True if the string is an alphabetic string, False otherwise.
|
|      A string is alphabetic if all characters in the string are alphabetic and th
ere
|      is at least one character in the string.
```

```
 |
 |  isascii(self, /)
 |      Return True if all characters in the string are ASCII, False otherwise.
 |
 |      ASCII characters have code points in the range U+0000-U+007F.
 |      Empty string is ASCII too.
 |
 |  isdecimal(self, /)
 |      Return True if the string is a decimal string, False otherwise.
 |
 |      A string is a decimal string if all characters in the string are decimal and
 |      there is at least one character in the string.
 |
 |  isdigit(self, /)
 |      Return True if the string is a digit string, False otherwise.
 |
 |      A string is a digit string if all characters in the string are digits and th
ere
 |      is at least one character in the string.
 |
 |  isidentifier(self, /)
 |      Return True if the string is a valid Python identifier, False otherwise.
 |
 |      Call keyword.iskeyword(s) to test whether string s is a reserved identifier,
 |      such as "def" or "class".
 |
 |  islower(self, /)
 |      Return True if the string is a lowercase string, False otherwise.
 |
 |      A string is lowercase if all cased characters in the string are lowercase an
d
 |      there is at least one cased character in the string.
 |
 |  isnumeric(self, /)
 |      Return True if the string is a numeric string, False otherwise.
 |
 |      A string is numeric if all characters in the string are numeric and there is
at
 |      least one character in the string.
 |
 |  isprintable(self, /)
 |      Return True if the string is printable, False otherwise.
 |
 |      A string is printable if all of its characters are considered printable in
 |      repr() or if it is empty.
 |
 |  isspace(self, /)
 |      Return True if the string is a whitespace string, False otherwise.
 |
 |      A string is whitespace if all characters in the string are whitespace and th
ere
 |      is at least one character in the string.
 |
 |  istitle(self, /)
 |      Return True if the string is a title-cased string, False otherwise.
 |
 |      In a title-cased string, upper- and title-case characters may only
 |      follow uncased characters and lowercase characters only cased ones.
 |
 |  isupper(self, /)
 |      Return True if the string is an uppercase string, False otherwise.
 |
 |      A string is uppercase if all cased characters in the string are uppercase an
d
 |      there is at least one cased character in the string.
```

```
 |
 |  join(self, iterable, /)
 |      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'
 |
 |  ljust(self, width, fillchar=' ', /)
 |      Return a left-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  lower(self, /)
 |      Return a copy of the string converted to lowercase.
 |
 |  lstrip(self, chars=None, /)
 |      Return a copy of the string with leading whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  partition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string.  If the separator is foun
d,
 |      returns a 3-tuple containing the part before the separator, the separator
 |      itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing the original str
ing
 |      and two empty strings.
 |
 |  removeprefix(self, prefix, /)
 |      Return a str with the given prefix string removed if present.
 |
 |      If the string starts with the prefix string, return string[len(prefix):].
 |      Otherwise, return a copy of the original string.
 |
 |  removesuffix(self, suffix, /)
 |      Return a str with the given suffix string removed if present.
 |
 |      If the string ends with the suffix string and that suffix is not empty,
 |      return string[:-len(suffix)]. Otherwise, return a copy of the original
 |      string.
 |
 |  replace(self, old, new, count=-1, /)
 |      Return a copy with all occurrences of substring old replaced by new.
 |
 |        count
 |          Maximum number of occurrences to replace.
 |          -1 (the default value) means replace all occurrences.
 |
 |      If the optional argument count is given, only the first count occurrences ar
e
 |      replaced.
 |
 |  rfind(...)
 |      S.rfind(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
```

```
|      Return -1 on failure.
|
|  rindex(...)
|      S.rindex(sub[, start[, end]]) -> int
|
|      Return the highest index in S where substring sub is found,
|      such that sub is contained within S[start:end].  Optional
|      arguments start and end are interpreted as in slice notation.
|
|      Raises ValueError when the substring is not found.
|
|  rjust(self, width, fillchar=' ', /)
|      Return a right-justified string of length width.
|
|      Padding is done using the specified fill character (default is a space).
|
|  rpartition(self, sep, /)
|      Partition the string into three parts using the given separator.
|
|      This will search for the separator in the string, starting at the end. If
|      the separator is found, returns a 3-tuple containing the part before the
|      separator, the separator itself, and the part after it.
|
|      If the separator is not found, returns a 3-tuple containing two empty string
s
|      and the original string.
|
|  rsplit(self, /, sep=None, maxsplit=-1)
|      Return a list of the words in the string, using sep as the delimiter string.
|
|        sep
|          The delimiter according which to split the string.
|          None (the default value) means split according to any whitespace,
|          and discard empty strings from the result.
|        maxsplit
|          Maximum number of splits to do.
|          -1 (the default value) means no limit.
|
|      Splits are done starting at the end of the string and working to the front.
|
|  rstrip(self, chars=None, /)
|      Return a copy of the string with trailing whitespace removed.
|
|      If chars is given and not None, remove characters in chars instead.
|
|  split(self, /, sep=None, maxsplit=-1)
|      Return a list of the words in the string, using sep as the delimiter string.
|
|      sep
|        The delimiter according which to split the string.
|        None (the default value) means split according to any whitespace,
|        and discard empty strings from the result.
|      maxsplit
|        Maximum number of splits to do.
|        -1 (the default value) means no limit.
|
|  splitlines(self, /, keepends=False)
|      Return a list of the lines in the string, breaking at line boundaries.
|
|      Line breaks are not included in the resulting list unless keepends is given
and
|      true.
|
|  startswith(...)
|      S.startswith(prefix[, start[, end]]) -> bool
```

```
 |
 |      Return True if S starts with the specified prefix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(self, chars=None, /)
 |      Return a copy of the string with leading and trailing whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  swapcase(self, /)
 |      Convert uppercase characters to lowercase and lowercase characters to upperc
ase.
 |
 |  title(self, /)
 |      Return a version of the string where each word is titlecased.
 |
 |      More specifically, words start with uppercased characters and all remaining
 |      cased characters have lower case.
 |
 |  translate(self, table, /)
 |      Replace each character in the string using the given translation table.
 |
 |        table
 |          Translation table, which must be a mapping of Unicode ordinals to
 |          Unicode ordinals, strings, or None.
 |
 |      The table must implement lookup/indexing via __getitem__, for instance a
 |      dictionary or list.  If this operation raises LookupError, the character is
 |      left untouched.  Characters mapped to None are deleted.
 |
 |  upper(self, /)
 |      Return a copy of the string converted to uppercase.
 |
 |  zfill(self, width, /)
 |      Pad a numeric string with zeros on the left, to fill a field of the given wi
dth.
 |
 |      The string is never truncated.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  maketrans(...)
 |      Return a translation table usable for str.translate().
 |
 |      If there is only one argument, it must be a dictionary mapping Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings or None.
 |      Character keys will be then converted to ordinals.
 |      If there are two arguments, they must be strings of equal length, and
 |      in the resulting dictionary, each character in x will be mapped to the
 |      character at the same position in y. If there is a third argument, it
 |      must be a string, whose characters will be mapped to None in the result.
```

In [ ]:

```
help(s1.split)
```

```
Help on built-in function split:

split(sep=None, maxsplit=-1) method of builtins.str instance
```

```
Return a list of the words in the string, using sep as the delimiter string.

sep
  The delimiter according which to split the string.
  None (the default value) means split according to any whitespace,
  and discard empty strings from the result.
maxsplit
  Maximum number of splits to do.
  -1 (the default value) means no limit.
```

In [ ]:
```
s1
```

Out[ ]:
```
'Hello Python! I love you!'
```

In [ ]:
```
s1.split()
```

Out[ ]:
```
['Hello', 'Python!', 'I', 'love', 'you!']
```

In [ ]:
```
a = s1.split()
```

In [ ]:
```
a
```

Out[ ]:
```
['Hello', 'Python!', 'I', 'love', 'you!']
```

In [ ]:
```
s1.split('!')
```

Out[ ]:
```
['Hello Python', ' I love you', '']
```

In [ ]:
```
s1
```

Out[ ]:
```
'Hello Python! I love you!'
```

In [ ]:
```
s1.split('o')
```

Out[ ]:
```
['Hell', ' Pyth', 'n! I l', 've y', 'u!']
```

In [ ]:
```
"".join(['1','2','3'])
```

Out[ ]:
```
'123'
```

In [ ]:
```
s1
```

Out[ ]:
```
'Hello Python! I love you!'
```

In [ ]:
```
x = s1.split('o')
```

In [ ]:
```
x
```

Out[ ]:
```
['Hell', ' Pyth', 'n! I l', 've y', 'u!']
```

In [ ]:
```
"".join(x)
```

Out[ ]:
```
'Hell Pythn! I lve yu!'
```

In [ ]:
```
x = s1.split()
```

In [ ]:
```
x
```

Out[ ]:
```
['Hello', 'Python!', 'I', 'love', 'you!']
```

In [ ]:
```
" ".join(x)
```

Out[ ]:
```
'Hello Python! I love you!'
```

In [ ]:
```
a = input("enter: ")
a = a.split()
```

In [ ]:
```
a
```

Out[ ]:
```
['1', '2', '3', '4', '5', '6', '7']
```

In [ ]:
```
"".join(a)
```

Out[ ]:
```
'1234567'
```

In [ ]:
```
# task 11.1
a = [37, 0, 50, 46, 34]
```

In [ ]:
```
# task 11.2
a = []
i = 1
while i <= 10:
    s = int(input("Sandy engiz: "))
    a.append(s)
    i += 1

print(a)
```

```
[2, 2, 4, 5, 6, 7, 8, 9, 1, 2]
```

In [ ]:
```
# task 11.2
a = []
i = 1
while i <= 10:
    a.append(int(input("Sandy engiz: ")))
    i += 1

print(a)
```

```
[3, 3, 2, 1, 56, 7, 8, 23, 45, 56]
```

In [ ]:
```
# task 11.2
s = input("Sandy engiz: ")
a = s.split()
a = [int(i) for i in a]

# for i in a:
#     a.append(int(a))

print(a)
```

```
[3, 4, 5, 6, 3, 6, 7, 8, 9, 10]
```

In [ ]:
```
# task 11.3a
import random as rn
a  = []
for i in range(15):
    a.append(round(rn.random(), 3))
print(a)
```

```
[0.613, 0.993, 0.355, 0.003, 0.903, 0.542, 0.208, 0.474, 0.165, 0.312, 0.147, 0.703,
0.744, 0.162, 0.546]
```

In [ ]:
```
# task 11.3b
import random as rn
a  = []
for i in range(15):
    a.append(rn.randrange(10))
print(a)
```

```
[5, 8, 4, 2, 4, 0, 8, 0, 2, 1, 6, 8, 7, 4, 4]
```

In [ ]:
```
# task 12.1
first_name = input("First name: ")
second_name = input("Second name: ")
name  = first_name + second_name
print(name)
```

```
DauletBaiymbet
```

In [ ]:
```
# task 12.2
country = input("Country: ")
city = input("City: ")
print("The capital of the {0} is {1}".format(country, city))
print(f"The capital of the {country} is {city}")
print("The capital of the " + country + " is " +  city)
```

```
The capital of the Kazakhstan is Nur-Sultan
The capital of the Kazakhstan is Nur-Sultan
The capital of the Kazakhstan is Nur-Sultan
```

In [ ]:
```

```

Тапсырма:

Дмитрий Злотопольский 1400 задач по программированию кітабынан [11..12] бөлімдерінен 4 есептен шығарыңыздар. Әр бөлімнен кез-келген 4 есепті таңдайсыздар.