VTI Academy
WAY TO ENTERPRISE

# Basic Programming

## Lesson 03

# Outline

1. String
2. List
3. Tuple
4. Dictionary

# String

# String quoting styles

Literal strings in Python are delimited by quotes:

```
>>> 'This is a string'
```

Or you can use double quotation marks, as shown below:

```
>>> "This is also a string"
```

You must, however, be consistent:

```
>>> "inconsistent'
```

Supporting both quoting styles allows you to easily incorporate the other quote character into the literal strings:

```
>>> "It's a good thing."
"It's a good thing."
>>> ' "Yes!", he said, "I agree!" '
' "Yes!", he said, "I agree!" '
```

# Multiline strings and newlines

Use three quote characters:

```
>>> """This is
. . . a multiline
. . . string"""
```

Use three single-quotes:

```
>>> '''So
. . . is
. . . this.'''
```

Use the control characters:

```
>>> 'This string\nspans mutiple\nlines'
```

# Escape sequences

Use escape sequences for incorporating tabs with \t or using quote characters inside strings with \:

```
>>> "This is a \" in a string"
'This is a " in a string'
>>> 'This is a \' in a string'
"This is a ' in a string"
>>> 'This is a \" and a \' in a string'
'This is a " and a \' in a string'
>>> 'This is a \t in a string'
```

# Escape sequences

Because the backslash has special meaning, to place a backslash in a string we must escape the backslash with itself:

```
>>> k = 'A \\ in a string'
'A \\ in a string'
>>> print(k)
A \ in a string
```

To create a raw string, precede the opening quote with a lower-case r:

```
>>> path = r'C:\Users\Merlin\Documents\Spells'
'C:\\Users\\Merlin\\Documents\\Spells'
>>> print(path)
C:\Users\Merlin\Documents\Spells
```

# str constructor

Create strings representations of other types, such as integers:

```
>>> str(496)
'496'
```

Or floats:

```
>>> str(6.02e23)
'6.02e+23'
```

Or boolean:

```
>>> str(True)
'True'
```

# Strings as sequences

Access individual characters using square brackets with a zero-based integer index:

```
>>> s = 'parrot'
>>> s[4]
'o'
>>> s[2:4]
'rr'
>>> s[:4]
'parr'
```

```
>>> s[-1]
't'
>>> s[3:]
'rot'
```

# Check string

To check if a certain phrase or character is present in a string, we can use the keyword in:

```
>>> txt = "Welcome to Python Core!"
>>> print("Python" in txt)
True
>>> print("Python" not in txt)
False
>>> print("Programming" in txt)
False
>>> print("Programming" not in txt)
True
```

# String functions

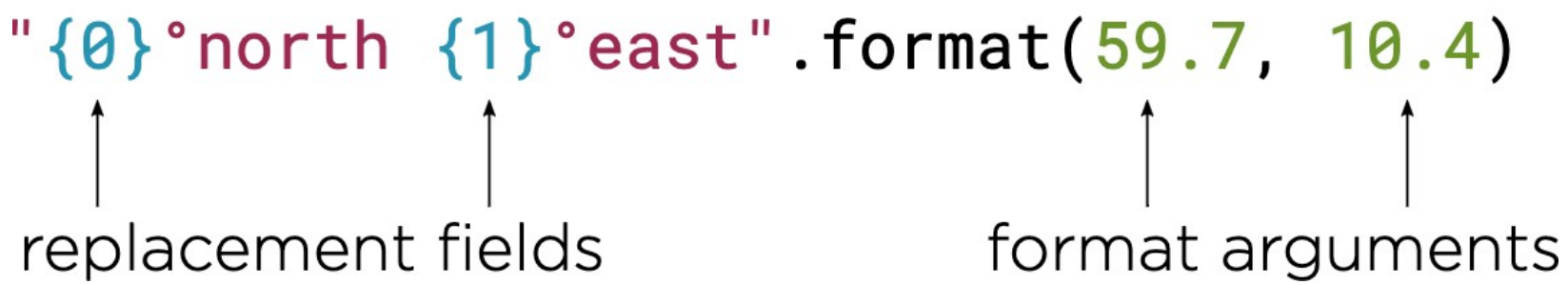
Python has built-in functions to help us process strings:

```
>>> print(txt.upper())
WELCOME TO PYTHON CORE!
>>> print(txt.lower())
welcome to python core!
>>> print(txt.split(' '))
['Welcome', 'to', 'Python', 'Core']
>>> print(txt.replace('Core', 'Language'))
Welcome to Python Language!
>>> ';'.join(['Welcome', 'to', 'VTI Academy'])
'Welcome;to;VTI Academy'
```

# String format

We can format the display string using the following syntax:

```
>>> subject = 'Python'
>>> print('I\'m %s' %(subject))
I’m Python
>>> f"one plus one is {1 + 1}"
```



# String format

Format Types:

| | |
|---|---|
| %c: | Single character |
| %s: | String |
| %i %d %u: | Signed integer decimal |
| %o: | Signed octal value |
| %x: | Signed hexadecimal lowercase |
| %X: | Signed hexadecimal uppercase |
| %e: | Floating point exponential format lowercase |
| %E: | Floating point exponential format uppercase |
| %f: | Floating point decimal format, etc. |

# List

# List – a sequence of objects

List of items are delimited by square brackets, and the items within the list are separated by commas. Here is a list of three numbers:

```
>>> [1, 9, 8]
[1, 9, 8]
```

And here is a list of three strings:

```
>>> ["apple", "orange", "pear"]
['apple', 'orange', 'pear']
```

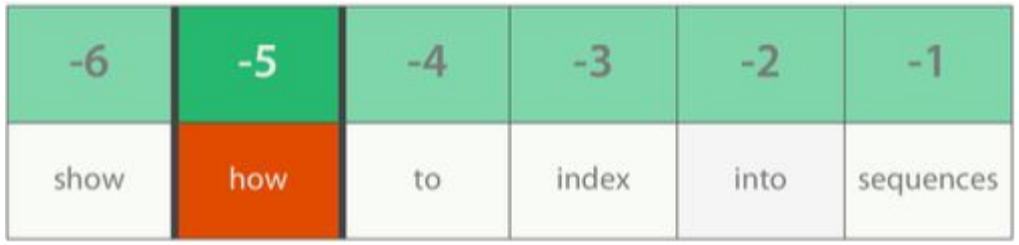
List of items can have different data types:

```
>>> [1, 3, 4, 'Python', True]
[1, 3, 4, 'Python', True]
```

# List indexing

Negative indexing for lists (and other sequences):

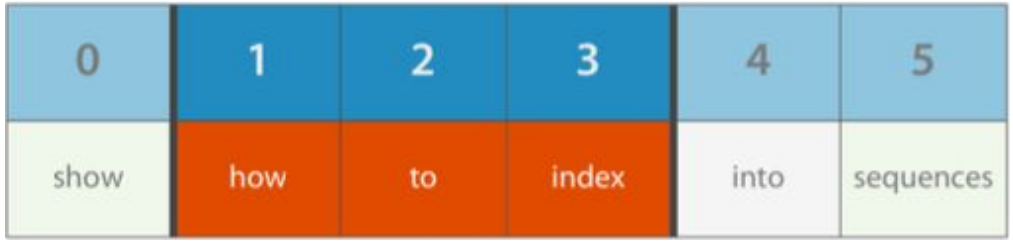
```
>>> r = ['show', 'how', 'to', 'index', 'into', 'sequences']
>>> r[-1]
'sequences'
>>> r[-5]
'how'
```



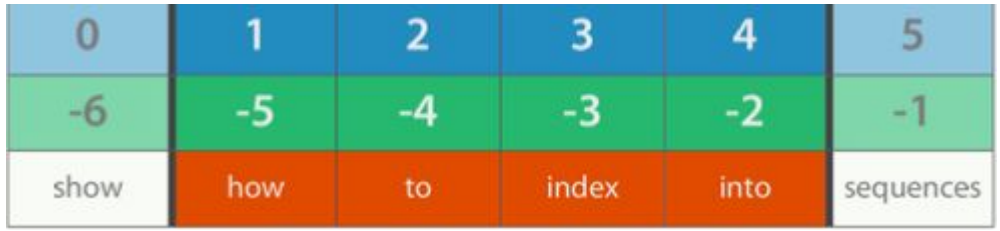
# List indexing

Slicing lists:

```
>>> r[1:4]
['how', 'to', 'index']
```




```
>>> r[1:-1]
['how', 'to', 'index', 'into']
```



# List indexing

Slicing lists:

```
>>> r[3:]
['index', 'into', 'sequences']
```


```
>>> r[:3]
['show', 'how', 'to']
```

# List indexing

Slicing lists:

```
>>> r[:]
['show', 'how', 'to', 'index', 'into', 'sequences']
```

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| -6 | -5 | -4 | -3 | -2 | -1 |
| show | how | to | index | into | sequences |

# Membership testing

To check if an item is exists in a list, we can use the keyword in:

```
>>> a = ["apple", "orange", "pear"]
>>> print("orange" in a)
True
>>> print("orange" not in a)
False
>>> print("Python" in a)
False
>>> print("Python" not in a)
True
```

# List in action

Reverse the elements of the list in place:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.reverse()
>>> print(numbers)
[1, 5, 8, 3]
```

Sort the items of the list in place:

```
>>> numbers.sort()
>>> print(numbers)
[1, 3, 5, 8]
>>> numbers.sort(reverse=True)
>>> print(numbers)
[8, 5, 3, 1]
```

# List in action

Add an item to the end of the list:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.append('Hello')
>>> print(numbers)
[3, 8, 5, 1, 'Hello']
```

Extend the list by appending all the items from the iterable:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.extend(['Python', 20.5])
>>> print(numbers)
[3, 8, 5, 1, 'Python', 20.5]
```

# List in action

Remove the first item from the list whose value is equal to x:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.remove(8)
>>> print(numbers)
[3, 5, 1]
```

Remove the item at the given position in the list, and return it:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.pop(2)
>>> print(numbers)
[3, 8, 1]
```

```
>>> numbers.pop()
>>> print(numbers)
[3, 8, 5]
```

# List in action

Remove all items from the list:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.clear()
>>> print(numbers)
[]
```

To destroy a list:

```
>>> numbers = [3, 8, 5, 1]
>>> del numbers
>>> print(numbers)
NameError: name 'numbers' is not defined
```

# List in action

Return a shallow copy of the list:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers_copy = numbers.copy()
# Or use: list(numbers), numbers[:]
>>> print(numbers_copy)
[3, 8, 5, 1]
>>> print(numbers_copy is numbers)
False
```

Return the number of times x appears in the list:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.count(8)
1
```

# List in action

Repeating lists:

```
>>> numbers = [3, 8, 5, 1]
>>> temp = numbers * 3
>>> print(temp)
[3, 8, 5, 1, 3, 8, 5, 1, 3, 8, 5, 1]
```

Return zero-based index in the list of the first item whose value is equal to x:

```
>>> numbers = [3, 8, 5, 1]
>>> i = numbers.index(8)
>>> print(i)
1
```

# List in action

Insert an item at a given position:

```
>>> numbers = [3, 8, 5, 1]
>>> numbers.insert(2, 'python')
>>> print(numbers)
[3, 8, 'python', 5, 1]
```

Concatenating lists:

```
>>> m = [2, 1, 3]
>>> n = [4, 7, 11]
>>> k = m + n
>>> print(k)
[2, 1, 3, 4, 7, 11]
```

# Tuple

# Tuple – an immutable sequence of objects

Tuples in Python are immutable sequences of arbitrary objects. Once created, the objects within them cannot be replaced or removed, and new elements cannot be added. Here is a tuple of three numbers:

```
>>> (1, 9, 8)
(1, 9, 8)
```

And here is a tuple of three strings:

```
>>> a = ("apple", "orange", "pear")
>>> a
('apple', 'orange', 'pear')
```

Tuple of items can have different data types:

```
>>> (1, 3, 4, 'Python', True)
(1, 3, 4, 'Python', True)
```

# Tuple element access

We can access the elements of a tuple by zero-based index using square brackets:

```
>>> t = ('Norway', 4.953, 3)
>>> t[0]
'Norway'
>>> t[-1]
3
```

Slicing tuples:

```
>>> t[:2]
('Norway', 4.953)
>>> t[:]
('Norway', 4.953, 3)
```

# Tuple in action

Repeating tuples:

```
>>> t = ('Norway', 4.953, 3)
>>> temp = t * 3
>>> print(temp)
('Norway', 4.953, 3, 'Norway', 4.953, 3, 'Norway', 4.953, 3)
```

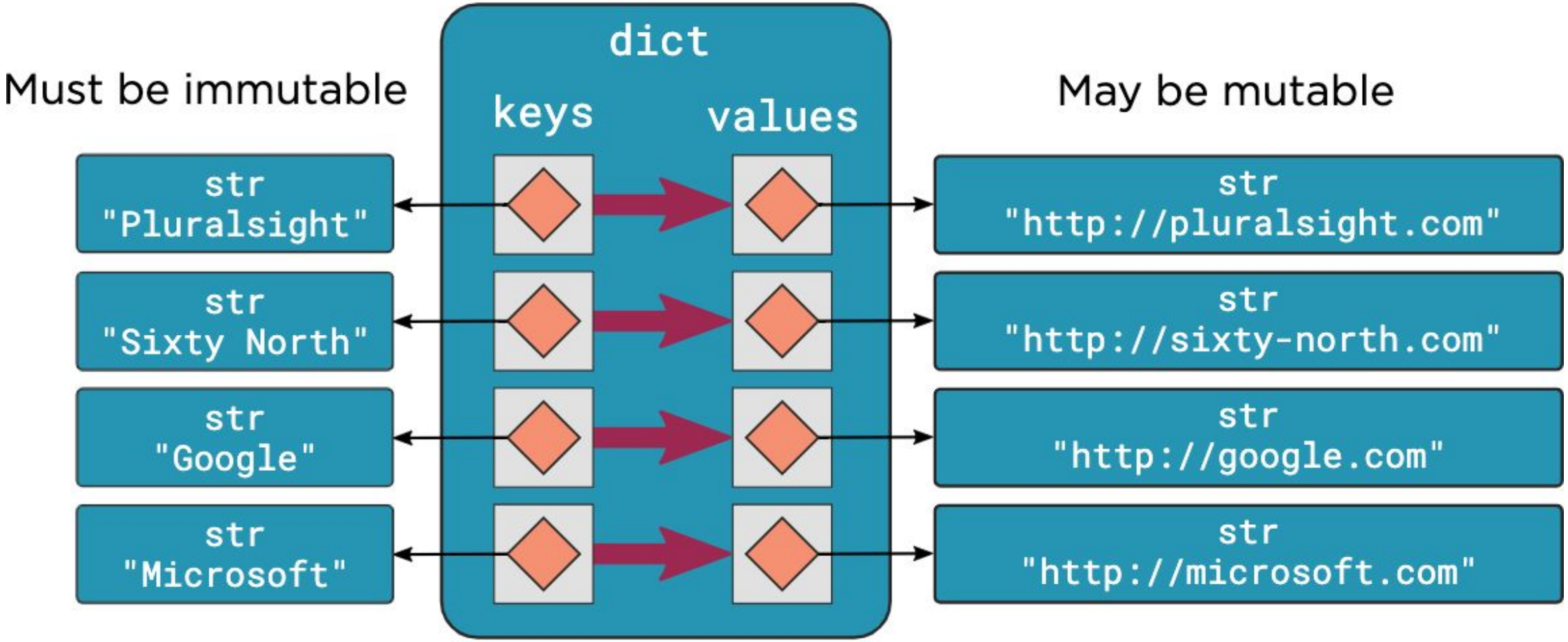
Concatenating tuples:

```
>>> temp = t + (338186.0, 265E9)
>>> print(temp)
('Norway', 4.953, 3, 338186.0, 265000000000.0)
```

# Dictionary

```
>>> urls = { 'Google': 'http://google.com',
...           'Pluralsight': 'http://pluralsight.com',
...           'Sixty North': 'http://sixty-north.com',
...           'Microsoft': 'http://microsoft.com' }
>>>
>>> urls['Pluralsight']
'http://pluralsight.com'
>>>
```

# Dictionary Internals




Do not rely on the order of items in the dictionary

```
>>> names_and_ages = [ ('Alice', 32), ('Bob', 48), ('Charlie', 28), ('Daniel', 3
3) ]
>>> d = dict(names_and_ages)
>>> d
{'Alice': 32, 'Bob': 48, 'Charlie': 28, 'Daniel': 33}
>>> phonetic = dict(a='alfa', b='bravo', c='charlie', d='delta', e='echo', f='fo
xtrot')
>>> phonetic
{'a': 'alfa', 'b': 'bravo', 'c': 'charlie', 'd': 'delta', 'e': 'echo', 'f': 'fox
trot'}
>>>
```

As with lists, dictionary copying is shallow.

```
>>> d = dict(goldenrod=0xDAA520, indigo=0x4B0082, seashell=0xFFF5EE)
>>> e = d.copy()
>>> e
{'goldenrod': 14329120, 'indigo': 4915330, 'seashell': 16774638}
>>> f = dict(e)
>>> f
{'goldenrod': 14329120, 'indigo': 4915330, 'seashell': 16774638}
>>> g = dict(wheat=0xF5DEB3, khaki=0xF0E68C, crimson=0xDC143C)
>>> f.update(g)
>>> f
{'goldenrod': 14329120, 'indigo': 4915330, 'seashell': 16774638, 'wheat': 161133
31, 'khaki': 15787660, 'crimson': 14423100}
>>> stocks = {'GOOG': 891, 'AAPL': 416, 'IBM': 194}
>>> stocks.update({'GOOG': 894, 'YHOO': 25})
>>> stocks
{'GOOG': 894, 'AAPL': 416, 'IBM': 194, 'YHOO': 25}
>>>
```

# dict.update()

Adds entries from one dictionary into another.

Call this on the dictionary that is to be updated.

# Dictionary iteration

Dictionaries yield the next key on each iteration.

Values can be retrieved using the square-bracket operator.

```
#A0522D
>>> for key in colors.keys():
...     print(key)
...
aquamarine
burlywood
chartreuse
cornflower
firebrick
honeydew
maroon
sienna
>>> for key, value in colors.items():
...     print(f"{key} => {value}")
...
aquamarine => #7FFFD4
burlywood => #DEB887
chartreuse => #7FFF00
cornflower => #6495ED
firebrick => #B22222
honeydew => #F0FFF0
maroon => #B03060
sienna => #A0522D
>>>
```

# `dict.items()`

Iterates over keys and values in tandem.

Yields a `(key, value)` tuple on each iteration.

```
>>> del z['Fy']
>>> z
{'H': 1, 'Tc': 43, 'Xe': 54, 'Rf': 104, 'Fm': 100}
>>> m = {'H': [1, 2, 3],
...      'He': [3, 4],
...      'Li': [6, 7],
...      'Be': [7, 9, 10],
...      'B': [10, 11],
...      'C': [11, 12, 13, 14]}
>>> m['H'] += [4, 5, 6, 7]
>>> m
{'H': [1, 2, 3, 4, 5, 6, 7], 'He': [3, 4], 'Li': [6, 7], 'Be': [7, 9, 10], 'B':
[10, 11], 'C': [11, 12, 13, 14]}
>>> m['N'] = [13, 14, 15]
>>> from pprint import pprint as pp
>>> pp(m)
{'B': [10, 11],
 'Be': [7, 9, 10],
 'C': [11, 12, 13, 14],
 'H': [1, 2, 3, 4, 5, 6, 7],
 'He': [3, 4],
 'Li': [6, 7],
 'N': [13, 14, 15]}
>>>
```