Basic Programming

Lesson 04

Functions

Copyright © VuDuyKhuong All Right Reserved

Defining Functions

```
>>> def square(x):
...     return x * x
...
>>> square(5)
25
>>> def launch_missiles():
...     print("Missiles launched!")
...
>>> launch_missiles()
Missiles launched!
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Early Return

```
>>> def even_or_odd(n):
...     if n % 2 == 0:
...         print("even")
...         return
...     print("odd")
...
>>> w = even_or_odd(31)
odd
>>> w is None
True
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Naming Special Functions

__feature__

Hard to prounounce!

Copyright © VuDuyKhuong All Right Reserved

dunder

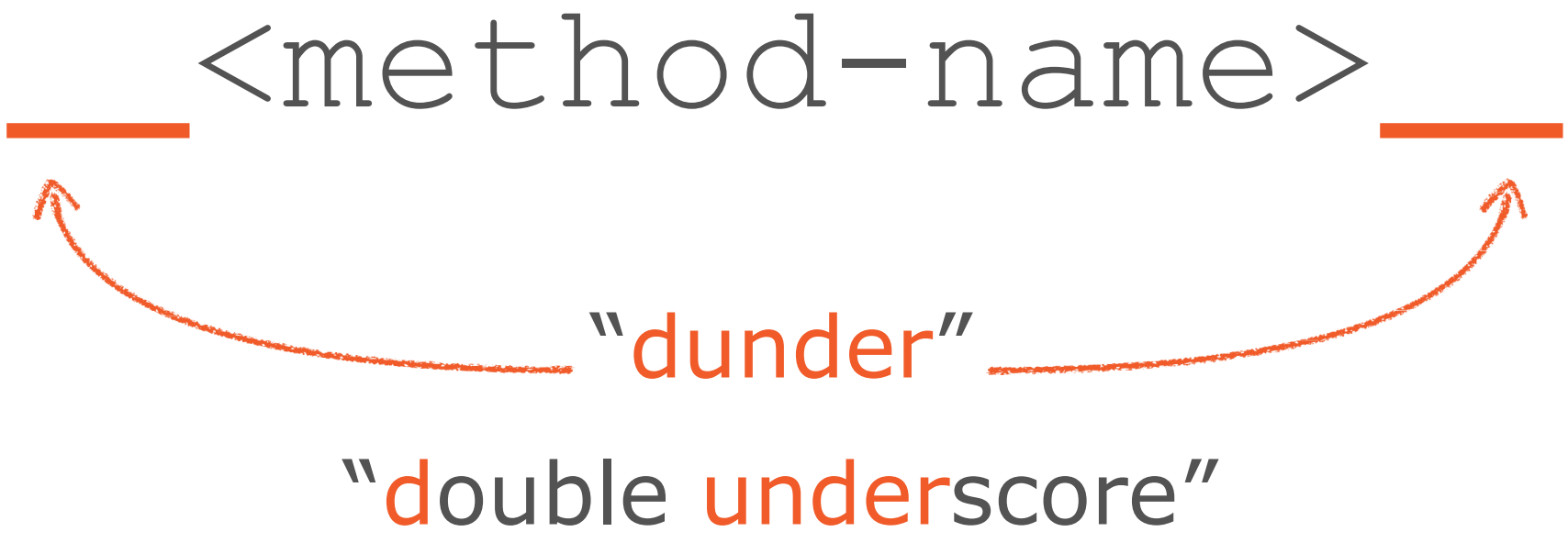
Our way of pronouncing special names

A portmanteau of "double underscore"

Instead of "underscore underscore name underscore underscore" we'll say "dunder name"

Copyright © VuDuyKhuong All Right Reserved

Terminology for Python Special Methods



Copyright © VuDuyKhuong All Right Reserved

Terminology for Python Special Methods

`__len__`

"dunder len"

Copyright © VuDuyKhuong All Right Reserved

Import or Execute

```
from urllib.request import urlopen

def fetch_words():
  story = urlopen('http://sixty-north.com/c/t.txt')
  story_words = []
  for line in story:
      line_words = line.decode('utf8').split()
      for word in line_words:
          story_words.append(word)
  story.close()

  for word in story_words:
      print(word)

if __name__ == '__main__':
    fetch_words()
```

Copyright © VuDuyKhuong All Right Reserved

Docstrings

Copyright © VuDuyKhuong All Right Reserved

docstrings

Literal strings which document functions, modules, and classes.

They must be the first statement in the blocks for these constructs.

Copyright © VuDuyKhuong All Right Reserved

Docstrings

```
>>> from words import *
>>> help(fetch_words)
Help on function fetch_words in module words:

fetch_words(url)
    Fetch a list of words from a URL.

    Args:
        url: The URL of a UTF-8 text document.

    Returns:
        A list of strings containing the words from
        the document.
(END)
```

Copyright © VuDuyKhuong All Right Reserved

Docstrings

```
Help on module words:

NAME
    words - Retrieve and print words from a URL.

DESCRIPTION
    Usage:

       python3 words.py <URL>

FUNCTIONS
    fetch_words(url)
        Fetch a list of words from a URL.

        Args:
            url: The URL of a UTF-8 text document.

        Returns:
            A list of strings containing the words from
            the document.

    main(url)
        Print each word from a text document from at a URL.
:
```

Copyright © VuDuyKhuong All Right Reserved

Comments

Copyright © VuDuyKhuong All Right Reserved

Comments

Code is ideally clear enough without ancillary explanation

Sometimes you need to explain why your code is written as it is

Comments in Python start with # and extend to the end of the line

Copyright © VuDuyKhuong All Right Reserved

Passing arguments and returning values

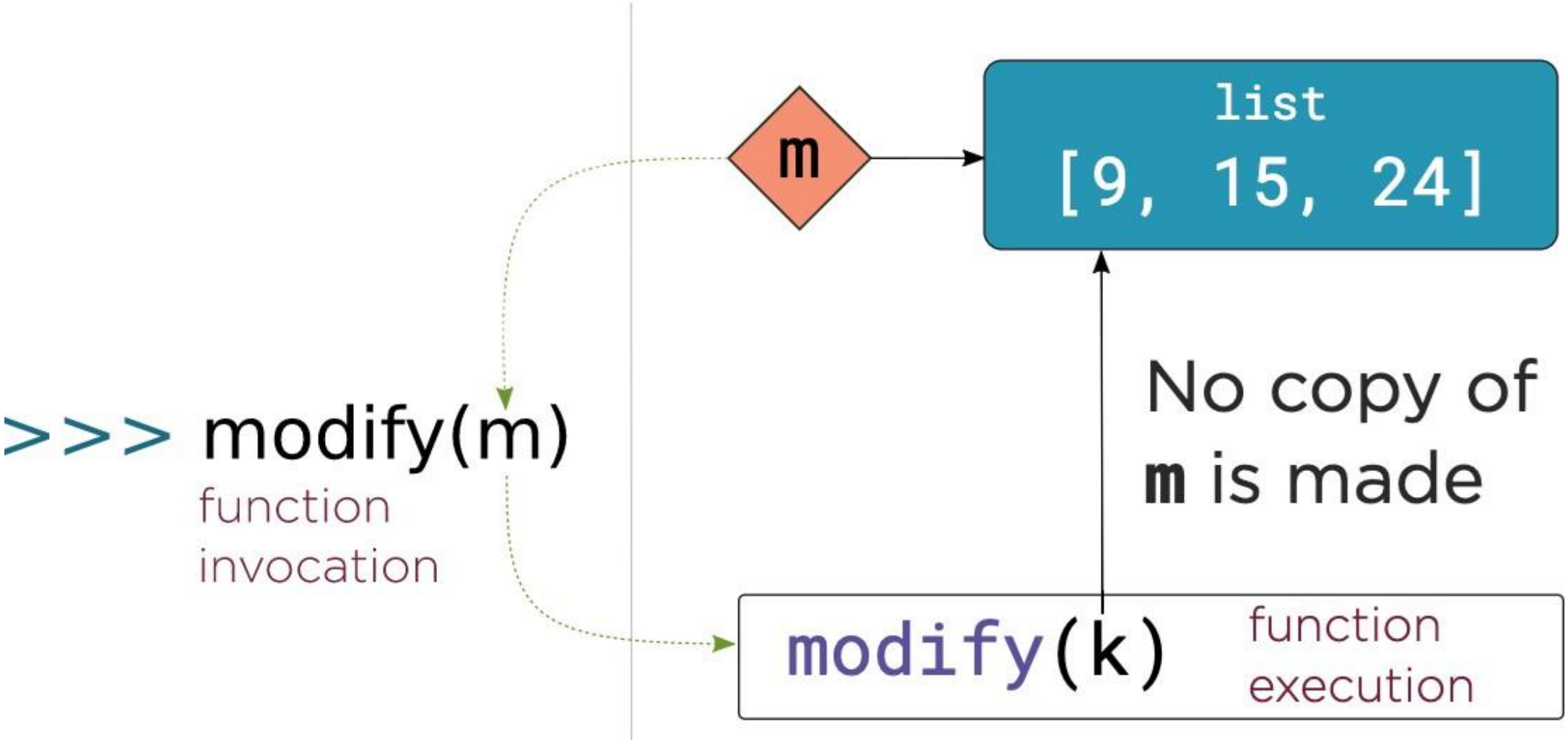
Copyright © VuDuyKhuong All Right Reserved

Argument Passing

```
>>> m = [9, 15, 24]
>>> def modify(k):
...         k.append(39)
...         print("k =", k)
...
>>> modify(m)
k = [9, 15, 24, 39]
>>> m
[9, 15, 24, 39]
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Argument Passing Semantics



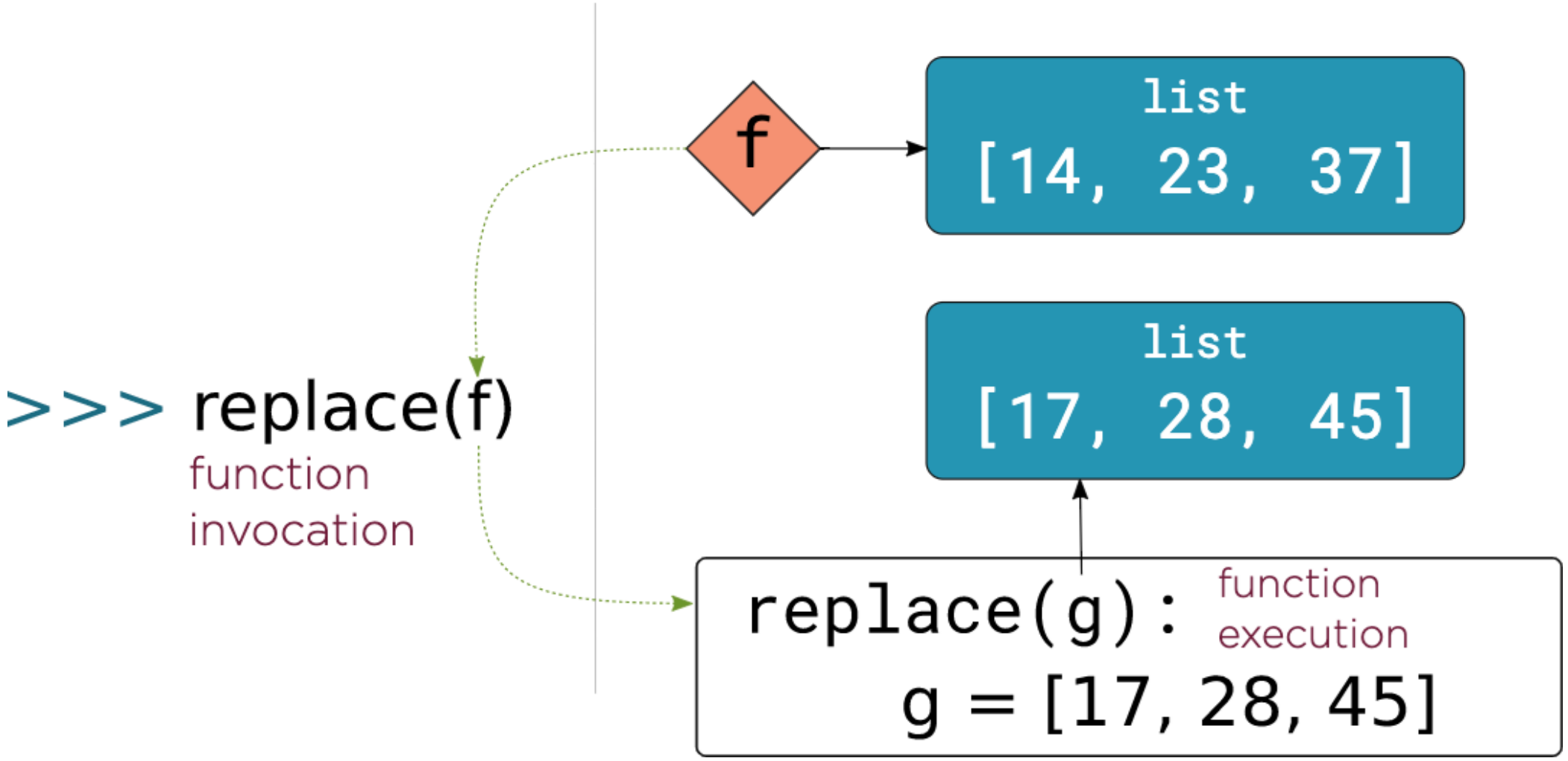
Copyright © VuDuyKhuong All Right Reserved

Replacing Argument Value

```
>>> f = [14, 23, 37]
>>> def replace(g):
...         g = [17, 28, 45]
...         print("g =", g)
...
>>> replace(f)
g = [17, 28, 45]
>>> f
[14, 23, 37]
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Replacing Argument Value



Copyright © VuDuyKhuong All Right Reserved

Return Semantics

```
>>> def f(d):
...     return d
...
>>> c = [6, 10, 16]
>>> e = f(c)
>>> c is e
True
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Function Arguments

Copyright © VuDuyKhuong All Right Reserved

Default Argument Values

```
>>> def banner(message, border='-'):
...     line = border * len(message)
...     print(line)
...     print(message)
...     print(line)
...
>>> banner("Norwegian Blue")
--------------
Norwegian Blue
--------------
>>> banner("Sun, Moon and Stars", "*")
******************
Sun, Moon and Stars
******************
>>> banner("Sun, Moon and Stars", border="*")
******************
Sun, Moon and Stars
******************
>>> banner(border=".", message="Hello from Earth")
................
Hello from Earth
................
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Arguments with default values must come after those without default values.

Copyright © VuDuyKhuong All Right Reserved

When are default values evaluated?

Copyright © VuDuyKhuong All Right Reserved

Default Value Evaluation

```
>>> import time
>>> time.ctime()
'Sun Nov 24 19:43:48 2019'
>>> def show_default(arg=time.ctime()):
...     print(arg)
...
>>> show_default()
Sun Nov 24 19:43:49 2019
>>> show_default()
Sun Nov 24 19:43:49 2019
>>> show_default()
Sun Nov 24 19:43:49 2019
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Default Value Evaluation

Remember that `def` is a statement executed at runtime.

Default arguments are evaluated when `def` is executed.

Immutable default values don't cause problems.

Mutable default values can cause confusing effects.

Copyright © VuDuyKhuong All Right Reserved

Always use immutable objects for default values.

Copyright © VuDuyKhuong All Right Reserved

Handle Exceptions

Copyright © VuDuyKhuong All Right Reserved

Exception handling

Mechanism for interrupting normal program flow and continuing in surrounding context

Copyright © VuDuyKhuong All Right Reserved

Exceptions: Key Concepts

1. Raising an exception
2. Handling an exception
3. Unhandled exceptions
4. Exception objects

Copyright © VuDuyKhuong All Right Reserved

Handle Exceptions

Cleanup Actions

Copyright © VuDuyKhuong All Right Reserved

try...finally

```
try:
    # try-block
finally:
    # executed no matter how the
    # try-block terminates
```

Copyright © VuDuyKhuong All Right Reserved

Not Exception-safe

```
import os


def make_at(path, dir_name):
    original_path = os.getcwd()
    os.chdir(path)
    os.mkdir(dir_name)
    os.chdir(original_path)
```

Copyright © VuDuyKhuong All Right Reserved

Handle Exception and Cleanup

```
import os
import sys

def make_at(path, dir_name):
    original_path = os.getcwd()
    os.chdir(path)
    try:
        os.mkdir(dir_name)
    except OSError as e:
        print(e, file=sys.stderr)
        raise
    finally:
        os.chdir(original_path)
```

Copyright © VuDuyKhuong All Right Reserved

Moment of Zen

Errors should never pass silently, unless explicitly silenced

Errors are like bells
And if we make them silent
They are of no use

Copyright © VuDuyKhuong All Right Reserved

Handle Exceptions

Exceptions and Control flow

Copyright © VuDuyKhuong All Right Reserved

```
DIGIT_MAP = {
    'zero':  '0',
    'one':   '1',
    'two':   '2',
    'three': '3',
    'four':  '4',
    'five':  '5',
    'six':   '6',
    'seven': '7',
    'eight': '8',
    'nine':  '9',
}

def convert(s):
    number = ''
    for token in s:
        number += DIGIT_MAP[token]
    x = int(number)
    return x
```

◀ Filename: exceptional.py

◀ Define a function

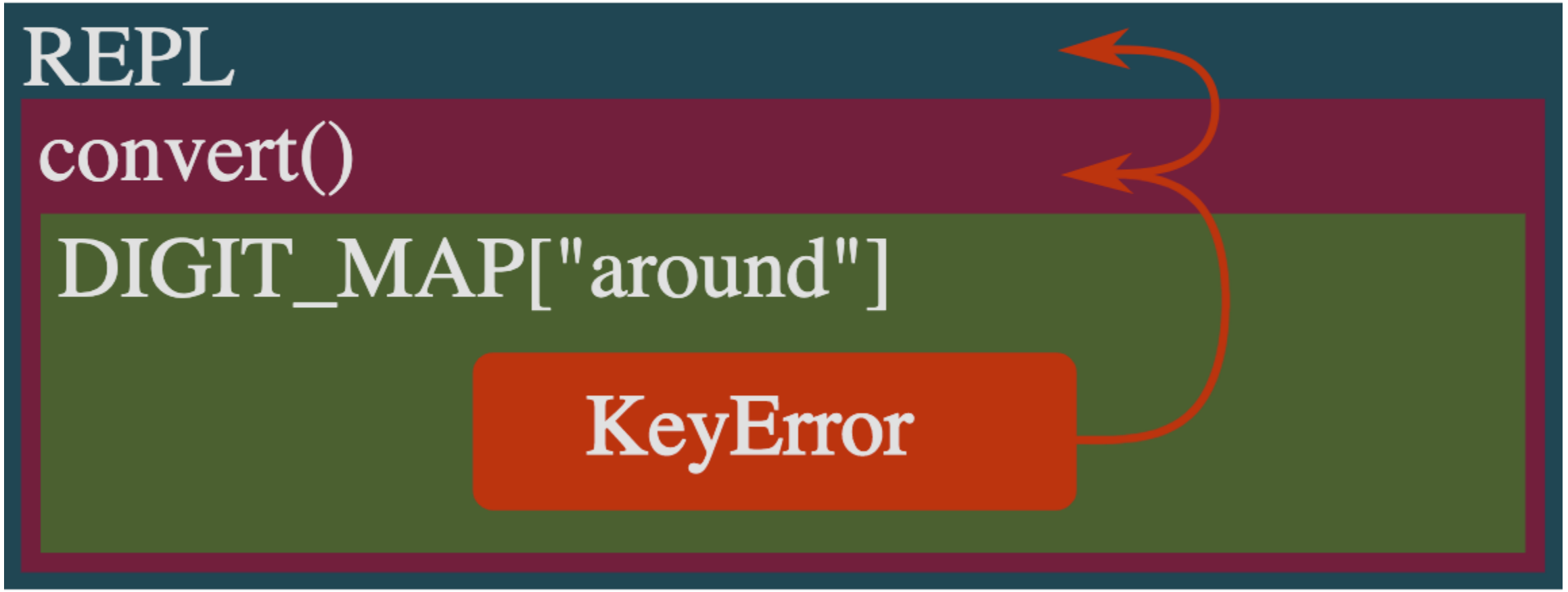
◀ Convert string to integer
◀ Return the integer

Copyright © VuDuyKhuong All Right Reserved

```
>>> from exceptional import convert
>>> convert("one three three seven".split())
1337
>>> convert("around two grillion".split())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/sixty-north/corepy/slide_spec/use-convert-v1/exceptional.py", lin
e 18, in convert
    number += DIGIT_MAP[token]
KeyError: 'around'
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Exception Propagation



Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        x = int(number)
    except KeyError:
        x = -1
    return x
```

- try-block
- Raise exceptions

- except-block
- Handle exceptions

Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        x = int(number)
        print(f"Conversion succeeded! x = {x}")
    except KeyError:
        print("Conversion failed!")
        x = -1
    return x
```

◀ Print on success

◀ Print on failure

Copyright © VuDuyKhuong All Right Reserved

```
>>> from exceptional import convert
>>> convert("three four".split())
Conversion succeeded! x = 34
34
>>> convert("eleventeen".split())
Conversion failed!
-1
>>> convert(512)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/sixty-north/corepy/slide_spec/use-convert-v3/exceptional.py", lin
e 18, in convert
    for token in s:
TypeError: 'int' object is not iterable
>>>
```

Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        x = int(number)
        print(f"Conversion succeeded! x = {x}")
    except KeyError:
        print("Conversion failed!")
        x = -1
    return x
```

◀ Not executed

◀ Executed

Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    """Convert a string to an integer."""
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        x = int(number)
        print(f"Conversion succeeded! x = {x}")
    except KeyError:
        print("Conversion failed!")
        x = -1
    except TypeError:
        print("Conversion failed!")
        x = -1
    return x
```

- Duplication

- Add `TypeError` handler
- Duplication

Copyright © VuDuyKhuong All Right Reserved

```
>>> from exceptional import convert
>>> convert(512)
Conversion failed!
-1
>>>
```

Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    """Convert a string to an integer."""
    x = -1
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        x = int(number)
        print(f"Conversion succeeded! x = {x}")
    except KeyError:
        print("Conversion failed!")
    except TypeError:
        print("Conversion failed!")
    return x
```

◀ Assignment

◀ Duplication

◀ Duplication

Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    """Convert a string to an integer."""
    x = -1
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        x = int(number)
        print(f"Conversion succeeded! x = {x}")
    except (KeyError, TypeError):
        print("Conversion failed!")
    return x
```

◀ Merge except blocks

Copyright © VuDuyKhuong All Right Reserved

```
>>> from exceptional import convert
>>> convert("two nine".split())
Conversion succeeded! x = 29
29
>>> convert("elephant".split())
Conversion failed!
-1
>>> convert(451)
Conversion failed!
-1
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Exceptions resulting from programmer errors:

```
IndentationError

SyntaxError

NameError
```

These should almost never be caught.

Copyright © VuDuyKhuong All Right Reserved

Handle Exceptions

Accessing Exception Objects

Copyright © VuDuyKhuong All Right Reserved

```python
import sys

DIGIT_MAP =  . . .

def convert(s):
    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        return int(number)
    except (KeyError, TypeError) as e:
        print(f"Conversion error: {e!r}",
              file=sys.stderr)
        return -1
```

Copyright © VuDuyKhuong All Right Reserved

```
>>> from exceptional import convert
>>> convert("fail".split())
Conversion error: KeyError('fail')
-1
>>>
```

Copyright © VuDuyKhuong All Right Reserved

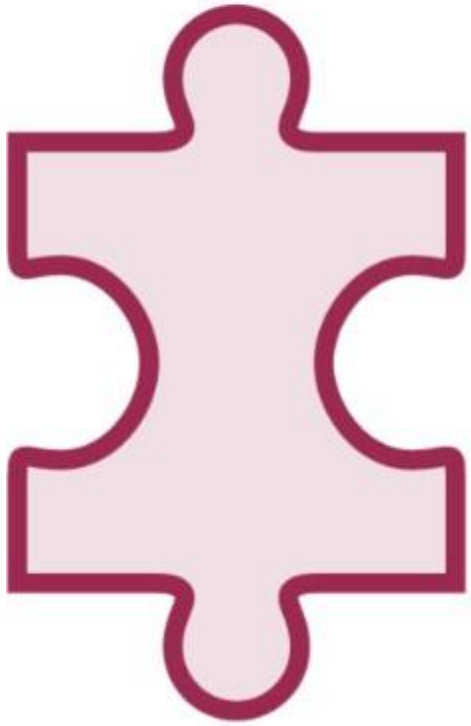
Exceptions Can Not Be Ignored

Error codes are easy to ignore

Checks are always required

Copyright © VuDuyKhuong All Right Reserved

Exceptions and Protocols



Sequences should raise `IndexError` for out-of-bounds indexing.

Exceptions must be implemented and documented correctly.

Existing built-in exceptions are often the right ones to use.

Copyright © VuDuyKhuong All Right Reserved

IndexError

An integer index is out of range

Copyright © VuDuyKhuong All Right Reserved

```
>>> z = [1, 4, 2]
>>> z[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

Copyright © VuDuyKhuong All Right Reserved

ValueError

An object is of the correct type but has an inappropriate value

Copyright © VuDuyKhuong All Right Reserved

```
>>> int("jim")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'jim'
>>>
```

Copyright © VuDuyKhuong All Right Reserved

KeyError

A lookup in a mapping failed

Copyright © VuDuyKhuong All Right Reserved

```
>>> codes = dict(gb=44, us=1, no=47, fr=33, es=34)
>>> codes['de']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'de'
>>>
```

Copyright © VuDuyKhuong All Right Reserved

Handle Exceptions

Avoid explicit type checks

Copyright © VuDuyKhuong All Right Reserved

```
def convert(s):
    if not isinstance(s, list):
        raise TypeError(
            "Argument must be a list")

    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        return int(number)
    except (KeyError, TypeError) as e:
        print(f"Conversion error: {e!r}",
              file=sys.stderr)
        raise
```

- Check argument type
- Raise TypeError

Copyright © VuDuyKhuong All Right Reserved

```python
def convert(s):
    # if not isinstance(s, list):
    #     raise TypeError(
    #         "Argument must be a list")

    try:
        number = ''
        for token in s:
            number += DIGIT_MAP[token]
        return int(number)
    except (KeyError, TypeError) as e:
        print(f"Conversion error: {e!r}",
              file=sys.stderr)
        raise
```

- Catch TypeError
- Re-raise it

Copyright © VuDuyKhuong All Right Reserved