

Recursion (Guttag 6)

and importing (Guttag 7)

Goals

Learn terms related to importing in python (Chapter 7)

Learn terms related to recursion (Chapter 6)

Practice three recursive algorithms (Chapter 6)

Importing

Guttag 7

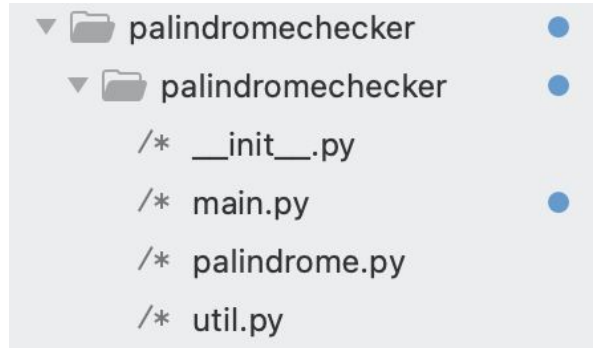
Terms: Module

Definition

- a module in python is a file that ends in .py

Examples from palindromes lab

- main.py
- util.py
- palidrome.py





Terms: Package


Definition

- a folder containing modules and an `__init__` module

Examples from palindromes lab

- innermost `palindromechecker` directory
- `tests` directory

```
▼  palindromechecker
  ▼  palindromechecker
    /* __init__.py
    /* main.py
    /* palindrome.py
    /* util.py
```

```
▼  tests
  /* __init__.py
  /* test_main.py
  /* test_palindrome.py
  /* test_util.py
```

mypackages

package_1

- __init__.py
- module_a.py
- module_b.py

package_2

- __init__.py
- module_a.py
- module_c.py

package_N

- __init__.py
- module_x.py
- module_y.py
- module_z.py

package_1

__init__.py

module_a.py

Classes

```
class MyClassA:  
    pass
```

```
class MyClassB:  
    pass
```

Functions

```
def myfunction_a():  
    pass
```

```
def myfunction_b():  
    pass
```

module_b.py

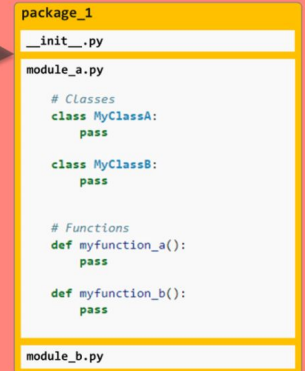
Terms: Symbol

Definition

- Anything defined within a module!
- recall, defined **variables** appear on the left-hand side of =
- recall, defined **functions** appear after keyword **def**
- we will also soon learn about **classes** appearing after keyword **class**

Examples

- **a** = 10
- **cli** = typer.Typer()
- def **is_prime**(n: int) -> bool:
- class **PalindromeCheckingApproach**(str, Enum):
- class **MyClassA**:
- class **MyClassB**:



```
package_1
__init__.py
module_a.py
    # Classes
    class MyClassA:
        pass
    class MyClassB:
        pass
    # Functions
    def myfunction_a():
        pass
    def myfunction_b():
        pass
module_b.py
```

Terms: Namespace & Fully-Qualified Name

Definition

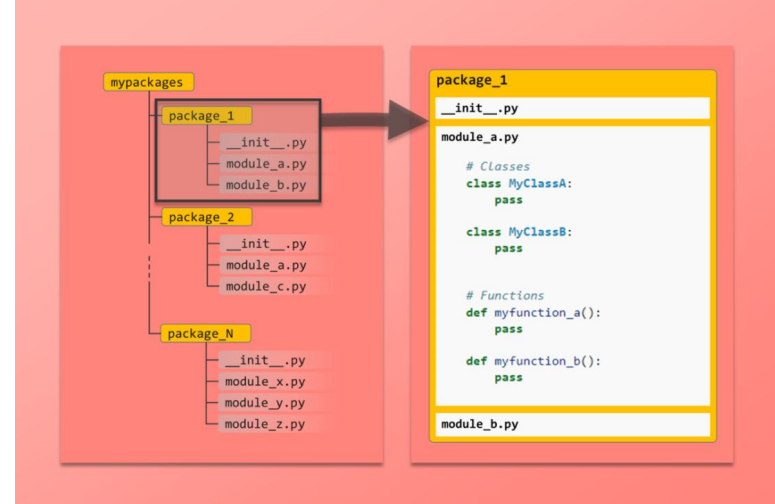
- Namespace refers to the module name
- Fully-Qualified Name specifies the Namespace and the symbol with dot notation

Examples

- `module_a.MyClassA`
- `module_a.MyClassB`

Further info

- Sometimes package names are included
- `package_1.module_a.MyClassA`



Terms: Import

Definition

- the python syntax used to "make available" symbols defined in different modules or packages.

Example

- `import typing`
- `import random`
- `import typer`

Further info

- The direct imports as shown above work for libraries, including the standard python libraries (included libraries)

Import Syntax

`import LIBRARY`

- `import typing`

`from LIBRARY import MODULE`

- `from typing import List`

`import MODULE as ALIAS`

- `import numpy as np`

Import Syntax Continued

from PACKAGE import MODULE

- from palindromechecker import util

from PACKAGE.MODULE import SYMBOL

- from palindromechecker.util import human_readable_boolean



Import Syntax Continued

from PACKAGE import MODULE

- from palindromechecker import util ← **util.human_readable_boolean ******

from PACKAGE.MODULE import SYMBOL

- from palindromechecker.util import human_readable_boolean



Recursion

Guttag 6

Definition

Applying the **same** logic repeatedly to solve a problem

- The problem **progresses** on each repetition

Problem-solving process stops when **base-case** is reached

Example: Factorial

informally:

$$5! = 5*4*3*2*1$$

formally:

$$1! = 1$$

$$n! = n * (n - 1)!$$

Example: Factorial

informally:

$$5! = 5*4*3*2*1$$

formally:

$$1! = 1$$

$$n! = n * (n - 1)!$$

Iterative Algorithm

```
def factorial_iter(n: int) ->  
int:
```

```
    result = 1
```

```
    for i in
```

```
        range(1,n+1,1):
```

```
            result *= i
```

```
    return result
```


Example: Factorial

informally:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

formally:

$$1! = 1$$

$$n! = n \cdot (n - 1)!$$

Iterative Algorithm

```
def factorial_iter(n: int) -> int:
```

```
    result = 1
```

```
    for i in  
        range(1, n+1, 1):
```

```
        result *= i
```

```
    return result
```

Recursive Algorithm

```
def factorial(n: int) -> int:
```

```
    if n == 1:
```

```
        return 1
```

```
    else:
```

```
        return n * factorial(n - 1)
```

Factorial: Recursive Approach, details

```
def factorial(n: int) -> int:
```

```
    if n == 1:
```

```
        return 1
```

base case (recursion stops)

```
    else:
```

```
        return n * factorial(n - 1)
```

recursive call with progression of the input

- `factorial` is the call
- `n-1` is progression of the input

Critical Thinking

What happens if:

- base case is missing?
 - infinite recursive calls
- there is no progression of the input?
 - infinite recursive calls
- progression is the wrong direction?
 - infinite recursive calls

Example: Palindrome

informally:

any string with any
characters that is
same forward and
backward

Example: Palindrome

informally:

any string with any
characters that is
same forward and
backward

Reverse Algorithm

```
def pal_rev(w: str) -> bool:  
    w_rev = w[::-1]  
    if w_rev == w:  
        return True  
    else:  
        return False
```

Example: Palindrome

informally:

any string with any characters that is same forward and backward

Reverse Algorithm

```
def pal_rev(w: str) -> bool:
    w_rev = w[::-1]
    if w_rev == w:
        return True
    else:
        return False
```

Recursive Algorithm

```
def pal_rec(w: str) -> bool:
    if len(w) <= 1:
        return True
    elif w[0] == w[-1]:
        return pal_rec(w[1:-1])
    else:
        return False
```

Palindrome: Recursive Approach, details

```
def pal_rec(w: str) -> bool:
```

```
    if len(w) <= 1:
```

base case (recursion stops)

```
        return True
```

```
    elif w[0] == w[-1]:
```

```
        return pal_rec(w[1:-1])
```

recursive call with progression of the input

- `pal_rec` is the call
- `w[1:-1]` is progression of the input

```
    else:
```

```
        return False
```

another base case (recursion stops)

Critical Thinking

Why does it matter:

- `if len(w) <= 1`
 - all strings that are one character or zero characters are palindromes!
- `w[0] == w[-1]`
 - the first and last character must be the same in a palindrome
- progression is done using `w[1:-1]`
 - slicing creates a shallow copied string starting at python 1st index and ending at -1 index (not inclusive!)
 - we already checked that the first character == last characters, thus we have to remove them to continue checking.

Example: Fibonacci Number

informally:

sum of previous two
numbers in a
sequence starting
with 0 and 1

Example: Fibonacci Number

informally:

sum of previous two
numbers in a
sequence starting
with 0 and 1

Iterative Algorithm

```
def fib(nth: int) -> int:
    if nth <= 1:
        return nth
    zeroth = 0
    first = 1
    for _ in range(2, nth + 1):
        next = zeroth + first
        zeroth = first
        first = next
    return next
```

Example: Fibonacci Number

informally:

sum of previous two
numbers in a
sequence starting
with 0 and 1

Iterative Algorithm

```
def fib(nth: int) -> int:
    if nth <= 1:
        return nth
    zeroth = 0
    first = 1
    for _ in range(2, nth + 1):
        next = zeroth + first
        zeroth = first
        first = next
    return next
```

Recursive Algorithm

```
def fib(nth: int) -> int:
    if nth <= 1:
        return nth
    else:
        return fib(nth-1) + fib(nth-2)
```

Fibonacci: Recursive Approach, details

```
def fib(nth: int) -> int:
```

```
    if nth <= 1:
```

```
        return nth
```

base cases (recursion stops)

```
    else:
```

```
        return fib(nth-1) + fib(nth-2)
```

two recursive calls with progression of the input

- `fib` is the call
- `n-1` is progression of the input
- `n-2` is progression of the input