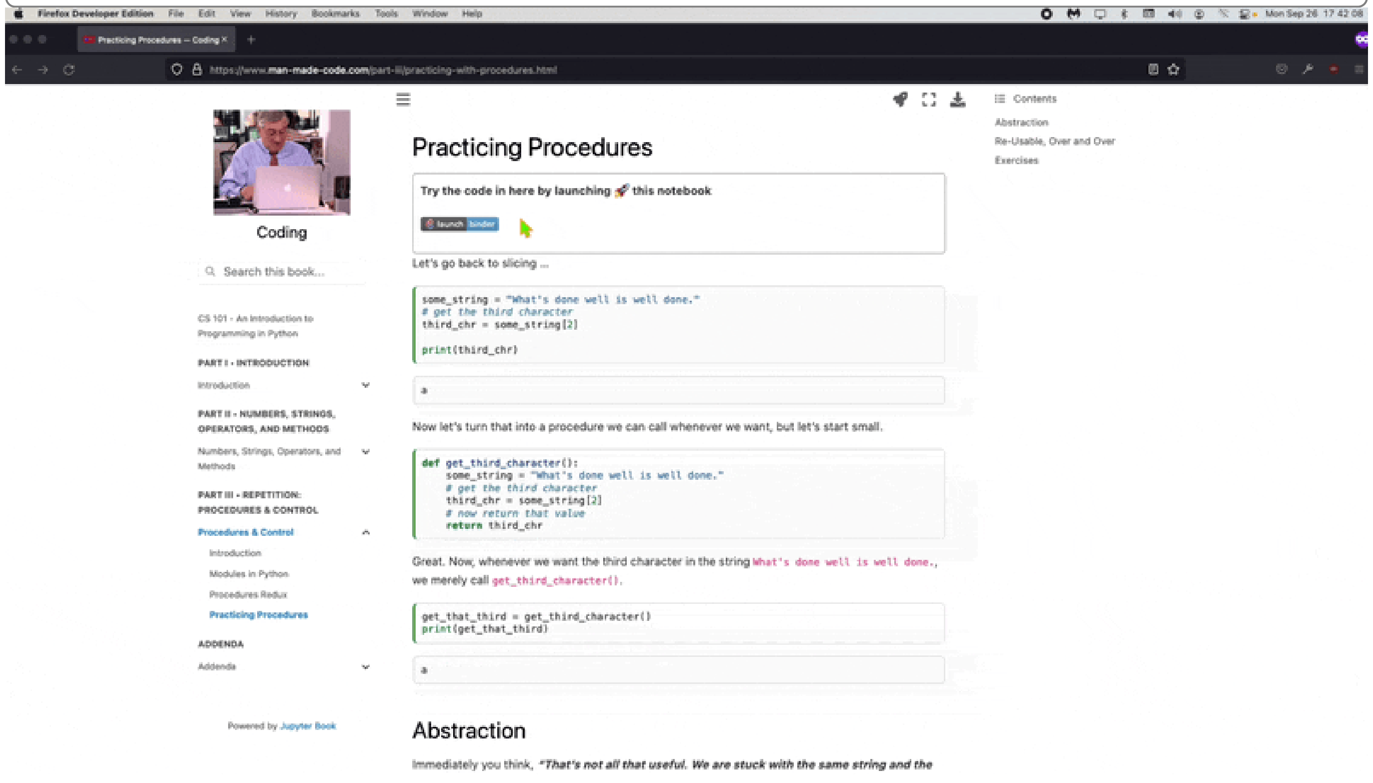
Practicing Procedures

Contents

Try the code in here by launching 🚀 this notebook

launch binder



Firefox Developer Edition File Edit View History Bookmarks Tools Window Help
Mon Sep 26 17 42 08
Practicing Procedures – Coding
https://www.man-made-code.com/part-iii/practicing-with-procedures.html
Coding
Search this book...
CS 101 - An Introduction to Programming in Python
PART I - INTRODUCTION
Introduction
PART II - NUMBERS, STRINGS, OPERATORS, AND METHODS
Numbers, Strings, Operators, and Methods
PART III - REPETITION: PROCEDURES & CONTROL
Procedures & Control
Introduction
Modules in Python
Procedures Redux
Practicing Procedures
ADDENDA
Addenda
Powered by Jupyter Book
Contents
Abstraction
Re-Usable, Over and Over
Exercises

Practicing Procedures

Try the code in here by launching 🚀 this notebook

launch binder

Let's go back to slicing ...

```
some_string = "What's done well is well done."
# get the third character
third_chr = some_string[2]

print(third_chr)
```

```
a
```

Now let's turn that into a procedure we can call whenever we want, but let's start small.

```
def get_third_character():
    some_string = "What's done well is well done."
    # get the third character
    third_chr = some_string[2]
    # now return that value
    return third_chr
```

Great. Now, whenever we want the third character in the string `What's done well is well done.`, we merely call `get_third_character()`.

```
get_that_third = get_third_character()
print(get_that_third)
```

```
a
```

Abstraction

Immediately you think, *"That's not all that useful. We are stuck with the same string and the*

Let's go back to slicing ...

```
some_string = "What's done well is well done."
# get the third character
third_chr = some_string[2]

print(third_chr)
```

```
a
```

Now let's turn that into a procedure we can call whenever we want, but let's start small.

```
def get_third_character():
    some_string = "What's done well is well done."
    # get the third character
    third_chr = some_string[2]
    # now return that value
    return third_chr
```

Great. Now, whenever we want the third character in the string `What's done well is well done.`, we merely call `get_third_character()`.

```
get_that_third = get_third_character()
print(get_that_third)
```

```
a
```

Abstraction

Immediately you think, ***"That's not all that useful. We are stuck with the same string and the same character every time. What's the point of such a constrained, limited procedure?"***

That's where `parameters` or `arguments` come into play. These are what give a procedure it's usefulness — it can act differently depending on the data we feed it.

And we ***feed it*** into the procedure by using those parentheses `()` found just before the colon in the procedure's definition.

So instead of being stuck with the same string and same character from that string, we'll alter the procedure.

And, in keeping with its new-found versatility, we'll rename it.

```
def get_a_char(chr_index_value):
    some_string = "What's done well is well done."
    character = some_string[chr_index_value]
    return character
```

Well, we're still stuck with the same string, but we've improved a bit: we can get any character from the string `What's done well is well done.`

```
ninth_chr = get_a_char(9)
print(ninth_chr)
```

```
n
```

A slight improvement — but let's improve it a little more: ***any*** string and any index value in that string (with appropriate nomenclature):

```
def get_chr(any_string, index):
    character = any_string[index]
    return character
```

Now we use any string and get the character occupying any index value in that string:

```
# string to use
our_string = "Four score and seven years ago"
# some integer
indx_value = 11
# call procedure and put in variable 'the_a'
the_a = get_chr(our_string, indx_value)

print(the_a)
```

```
a
```

Re-Usable, Over and Over

Keep in mind that, once you define the procedure, you can use it, i.e., `call` it over and over; moreover, you need not define it every time you call it. _**You define it once; you call it repeatedly.**

Exercises

Write a procedure that takes in three integers and returns the sum of their squares:

```python
def sum_of_squares(a, b, c):
    # note: remove the word pass below!
    # and replace it with your code
    pass
```

You can test your code if you launch this notebook in binder 👉 launch binder

Write a procedure that takes as arguments a string and returns a float representing half the string's length:

```python
def half_length(s):
    # note: remove the word pass below!
    # and replace it with your code
    pass
```

Write a procedure that takes in three characters (***single*** letter strings, e.g., 'a', 'r', '3') and return the sum of their Unicode code points. From the Python docs:

> Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, `ord('a')` returns the integer 97 … This is the inverse of `chr()`.

Thus:

```
>>> ord('a')
97
>>> ord('b')
98
>>> ord('4')
52
```

```python
def sum_code_points(x, y, z):
    # remember to replace pass with your code!
    pass
```

Write a procedure that increments a float or integer by 1 and returns that incremented value

```python
def incr(x):
    pass
```

Write a procedure that returns the very last character of a string

```python
def get_last_chr(a_string):
    pass
```

According to Newton's second law of motion, force can be expressed as the product of mass and acceleration of the body. Write a procedure that, given the mass and acceleration of an object, returns the force.

```python
def get_force(mass, acceleration):
    pass
```

Write a procedure that, taking in two strings, returns a string that is the first string, followed by two repetitions of the second string, followed by the first string.

```python
def abba_ize(a,b):
    pass
```

In case you missed it the first, seventh, and fifteenth times, try the code in here by launching 🚀 this notebook

launch binder