Announcements

- Labs begin this week
- Labs in computer labs (CSIT Building, Hanna Neumann Building)
  - Have a computer for every student
  - Also, can use your own computer
- Labs in Marie Reay Building are Bring-Your-Own-Device (BYOD) labs
  - Device == Laptop computer
  - Your device should have anaconda, python, etc installed
  - Charge your battery before attending the lab

Academic honesty

- Submitted code will be checked computationally for evidence of plagiarism.

- If evidence of plagiarism is found in individual homework problems, the mark for that individual homework will not be posted, until all homeworks have been assessed. In the context of all homeworks if it is decided there is evidence of repeated plagiarism, students will be interviewed for possible action of academic misconduct.

- The take-home assignment and exam will also be checked for evidence of academic misconduct.

- **What is okay**: for the homework, discussing the programming problems and approaches to solve them with other students is allowed, provided that no code is exchanged and that the final solution and code is written individually. In this case, the other students involved in the discussion must be listed in a comment at the top of the homework.
- For the final exam and take-home assignment must be individual work. You may not discuss the questions or your answers with anyone (this includes any on-line forum).

- Note that in all cases every line of code submitted must be fully written by you from scratch (and not just a modified copy of a version from the internet), And must be fully understood and explainable by you. Sufficient inline comments should be provided to make clear that you understand the code.

- Note on large language models and other code generators: generative AI models such as github copilot, chatGPT, Bing chatbot etc can be used by students for the homeworks and take-home assignment to explore solutions and understand their own code. They will not be allowed for the final exam. But in all cases the final code submitted by the student must be fully written and understood by the student, as described above.
- If you are unsure, please ask your tutor or the convenors.

Academic Honesty – Policy:

If we find evidence of cheating or copying of work, we will:

1. Review the potential copied work
2. Raise a flag on the student work as ‘under investigation’
3. Further evidence of cheating for a given student will result in all homework and/or assignment marks being set to 0 for that student

Fair use and acknowledgement

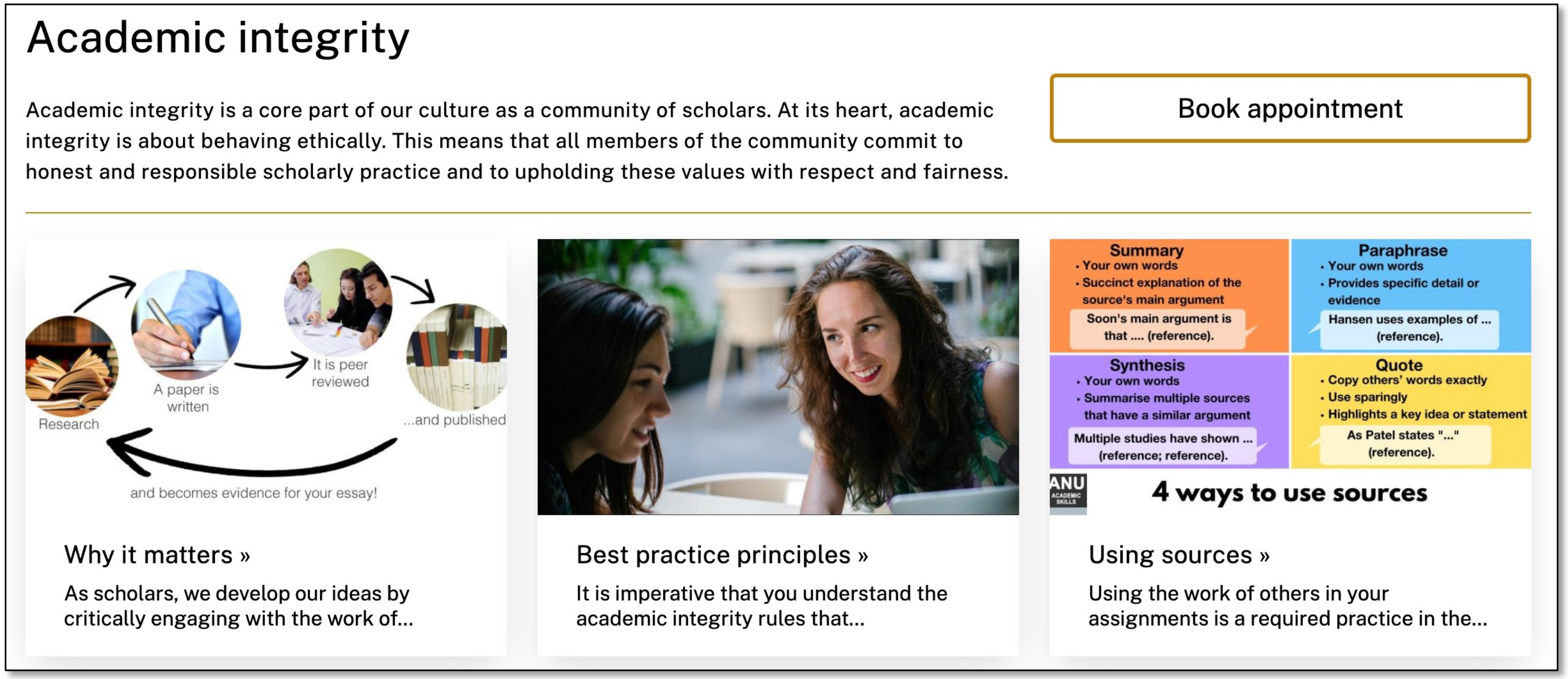
Australian National University

- ✗ Working on homework together and sharing the solution (even if you did half of it)
- ✗ Copying and pasting code or text from the internet (even if you changed it a bit)
- ✓ Talking to friends and classmates about questions
  - Acknowledge them by name at top of homework, just in case you make the same unusual error
- ✗ You let a friend read your completed homework or assignment
  - If your work looks enough like theirs during marking, you both will be punished equally
- ✗ ChatGPT/CoPilot wrote the first draft, but I completely re-wrote the whole document in my own words
  - Be careful – if you don't fully understand the solution, you will be marked very hard
  - If ChatGPT/CoPilot is wrong, you will look bad and you will be marked very hard
- ✓ You read programming books/websites and found examples where similar problems were solved and then wrote my own text/program with this knowledge (and you gave proper attribution to your sources).
  - Proper academic conduct requires you MUST cite your sources. It is not your primary work and you must give acknowledgement

Where to find more information

Australian National University

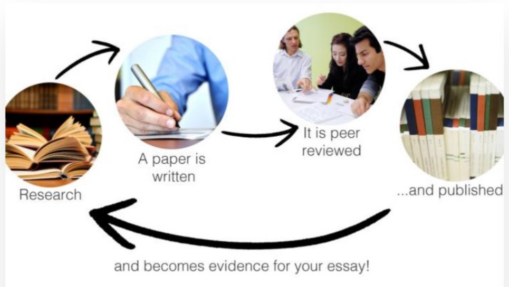
- Academic Skills @ ANU (https://www.anu.edu.au/students/academic-skills/academic-integrity)



Academic integrity

Academic integrity is a core part of our culture as a community of scholars. At its heart, academic integrity is about behaving ethically. This means that all members of the community commit to honest and responsible scholarly practice and to upholding these values with respect and fairness.

Book appointment

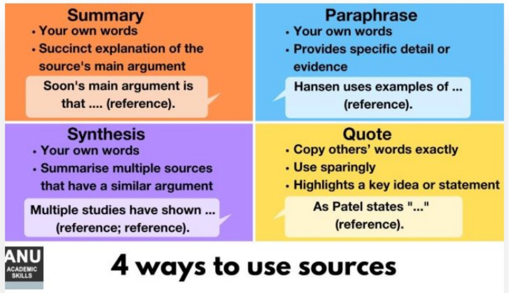


Why it matters »

As scholars, we develop our ideas by critically engaging with the work of...

Best practice principles »

It is imperative that you understand the academic integrity rules that...


Summary
• Your own words
• Succinct explanation of the source's main argument
Soon's main argument is that .... (reference).
Paraphrase
• Your own words
• Provides specific detail or evidence
Hansen uses examples of ... (reference).
Synthesis
• Your own words
• Summarise multiple sources that have a similar argument
Multiple studies have shown ... (reference; reference).
Quote
• Copy others' words exactly
• Use sparingly
• Highlights a key idea or statement
As Patel states "..." (reference).
ANU ACADEMIC SKILLS
4 ways to use sources

Using sources »

Using the work of others in your assignments is a required practice in the...

Assessment

Australian National University

- All assignment deadlines are hard – no late submissions will be accepted. Unless previous permission has been granted.
- Extension requests and late submissions **require documentary evidence**, such as a medical certificate
- Regarding deferred assessments and special consideration, please read: https://www.anu.edu.au/students/program-administration/assessments-exams
- Please note that "*any submitted work may be subject to an additional oral examination*", which can change the assessment mark in any way.

Lecture Roadmap

- Intro to Programming
- Variables
- Functions
    - The stack
    - Scope
- Functional abstraction
- Branching/Flow control
    - `if`
    - `while`
    - `for`
- Strings
- Lists
- Dictionaries

Functions

COMP1730/6730

Reading:

Chapter 3 : Downey, *Think Python*

Chapter 4: Sundnes, *Into to Sci Prog with Python*

OR

Sections 4.7 & 4.8: https://docs.python.org/3/tutorial/controlflow.html#defining-functions

Australian National University

Functions (*Think Python*, Ch. 3)

- Functions are like mini-programs you can call from your code that do useful, predefined tasks (that you would otherwise you might need to do for yourself)
- We have already seen a few:
  - `str(0.1)` converts a number (integer or float) to a string
  - `type(153)` prints the variable type
  - `print('this')` takes a string input and prints it to the terminal
- In python, functions can be:
  - Built-in
  - Imported from modules
  - User-defined

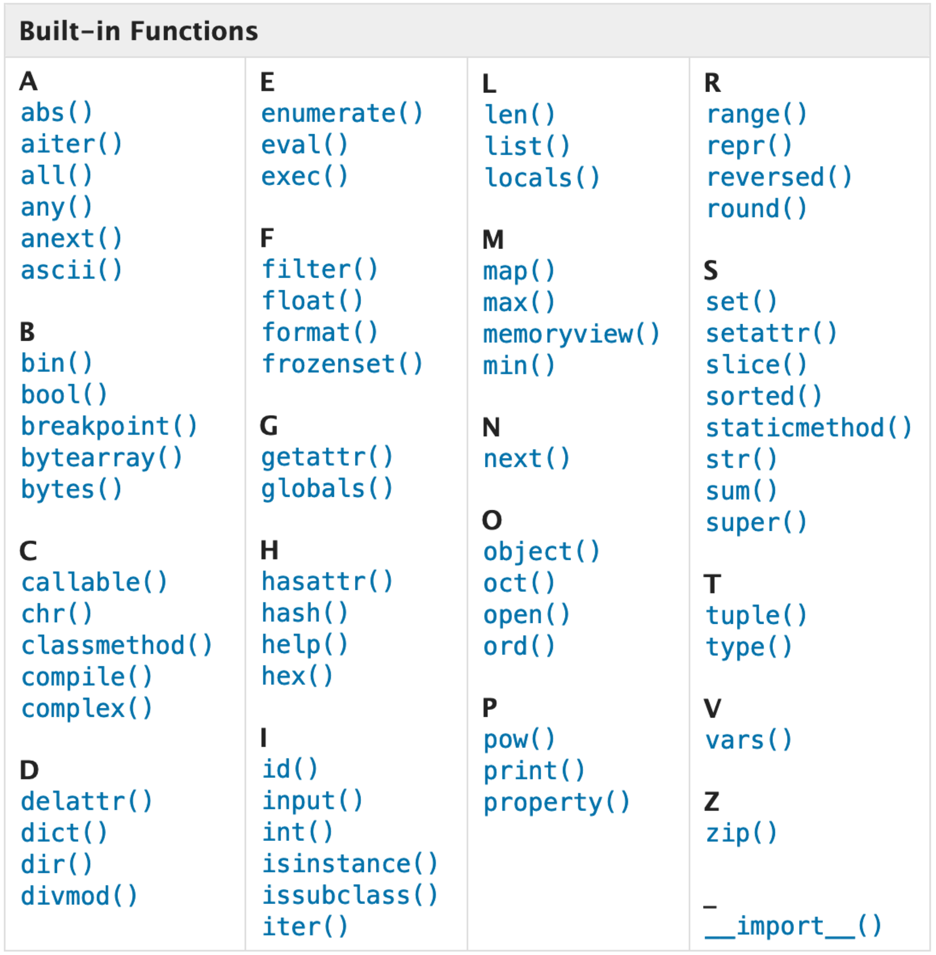
Built-in Python functions

Australian National University

- There LOTS of these. A very necessary part of the language
- Go to the source documentation for the Python language (at python.org) and look through what is available:

https://docs.python.org/3/library/functions.html

- Each function is described with details of what it does, what input it takes and what output it produces



Built-in Functions

A
abs()
aiter()
all()
any()
anext()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

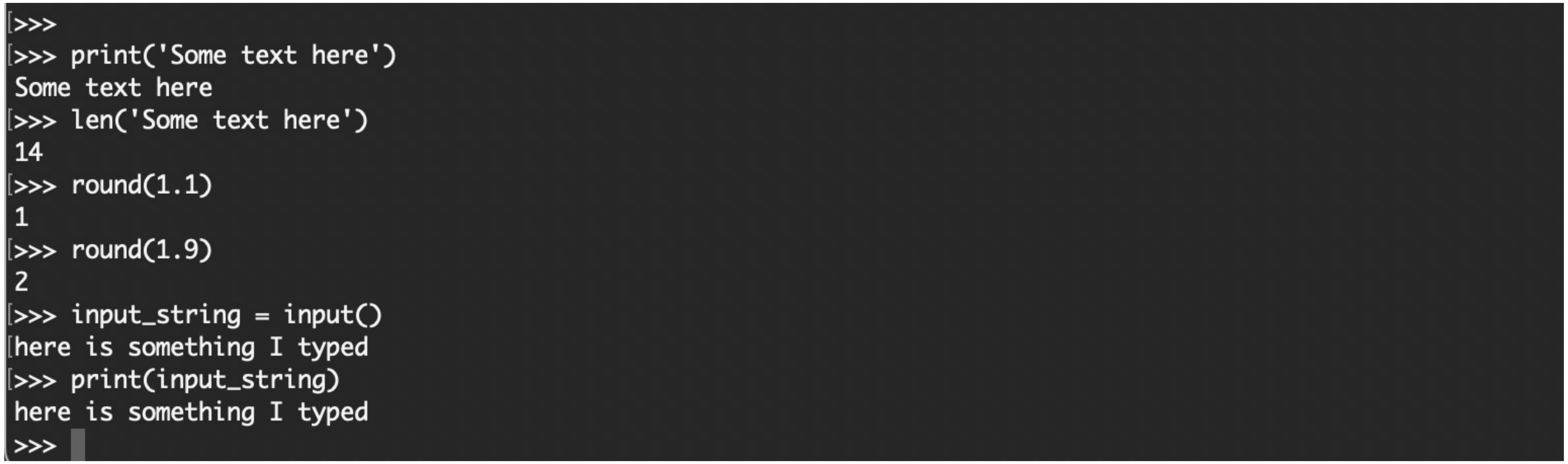
V
vars()

Z
zip()

_
__import__()

Example Built-in functions:
`print(), len(), round(), input()`

- As a useful exercise, go to the python.org documentation for built-in functions and look these up



```
[>>>
[>>> print('Some text here')
Some text here
[>>> len('Some text here')
14
[>>> round(1.1)
1
[>>> round(1.9)
2
[>>> input_string = input()
[here is something I typed
[>>> print(input_string)
here is something I typed
>>>
```

Example:

- Built-in functions:
  - print()
  - int()
  - len()
  - round()
  - abs()

- More at https://docs.python.org/3/library/functions.html

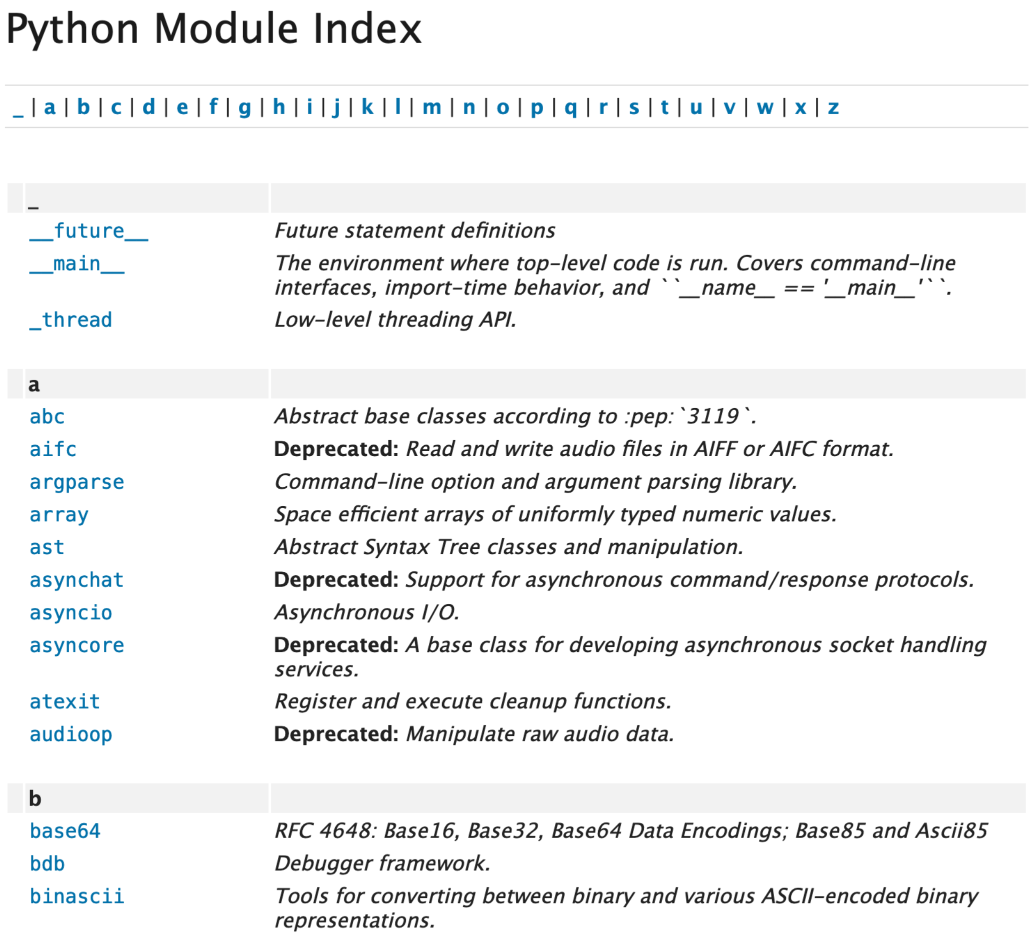
Imported Functions

- These work a lot like built-in functions, but need to be imported first and called with reference to the module they come from.
- Sometimes modules are referred to as packages or libraries
- A full list of Python modules is available at python.org:

https://docs.python.org/3/py-modindex.html

- Have a look at what is available – lots of useful stuff, eg:

math	json	pickle
random	statistics	getopt
zlib	pprint	csv



Python Module Index

_ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | z

_	
`__future__`	*Future statement definitions*
`__main__`	*The environment where top-level code is run. Covers command-line interfaces, import-time behavior, and ``__name__ == '__main__'``.*
`_thread`	*Low-level threading API.*
a	
`abc`	*Abstract base classes according to :pep:`3119`.*
`aifc`	**Deprecated:** *Read and write audio files in AIFF or AIFC format.*
`argparse`	*Command-line option and argument parsing library.*
`array`	*Space efficient arrays of uniformly typed numeric values.*
`ast`	*Abstract Syntax Tree classes and manipulation.*
`asynchat`	**Deprecated:** *Support for asynchronous command/response protocols.*
`asyncio`	*Asynchronous I/O.*
`asyncore`	**Deprecated:** *A base class for developing asynchronous socket handling services.*
`atexit`	*Register and execute cleanup functions.*
`audioop`	**Deprecated:** *Manipulate raw audio data.*
b	
`base64`	*RFC 4648: Base16, Base32, Base64 Data Encodings; Base85 and Ascii85*
`bdb`	*Debugger framework.*
`binascii`	*Tools for converting between binary and various ASCII-encoded binary representations.*

How to find out more

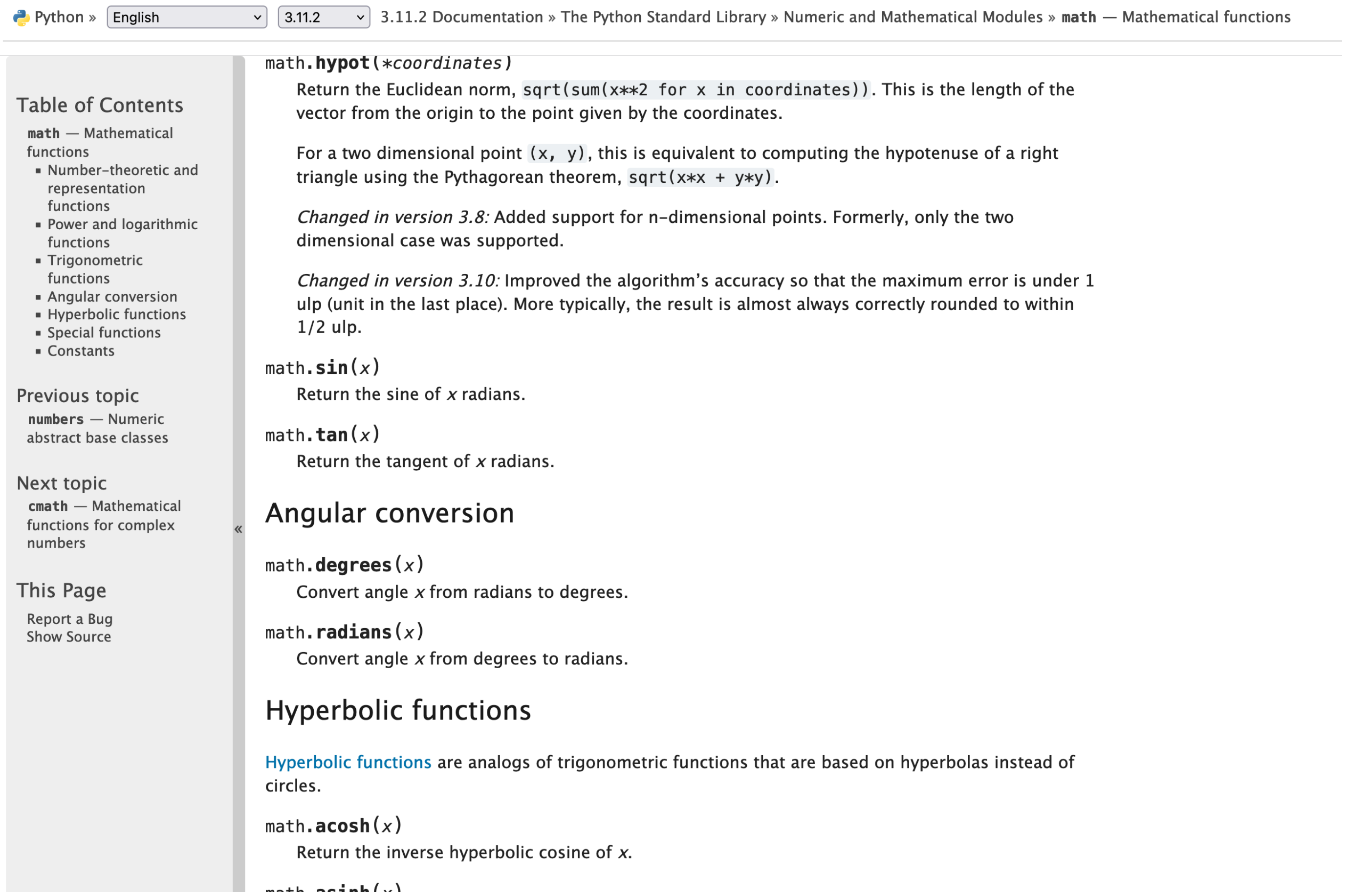
Australian National University

`docs.python.org` are your friend

This often the best way to find out how things work – the docs written by the developers.

If we scroll down this page far enough, the `math.sin()` is mentioned, as well as other useful things.

https://docs.python.org/3/library/math.html



Python » English 3.11.2 3.11.2 Documentation » The Python Standard Library » Numeric and Mathematical Modules » **math** — Mathematical functions

Table of Contents

math — Mathematical functions
- Number-theoretic and representation functions
- Power and logarithmic functions
- Trigonometric functions
- Angular conversion
- Hyperbolic functions
- Special functions
- Constants

Previous topic

numbers — Numeric abstract base classes

Next topic

cmath — Mathematical functions for complex numbers

This Page

Report a Bug
Show Source

`math.`**`hypot`**`(*coordinates)`

Return the Euclidean norm, `sqrt(sum(x**2 for x in coordinates))`. This is the length of the vector from the origin to the point given by the coordinates.

For a two dimensional point `(x, y)`, this is equivalent to computing the hypotenuse of a right triangle using the Pythagorean theorem, `sqrt(x*x + y*y)`.

Changed in version 3.8: Added support for n-dimensional points. Formerly, only the two dimensional case was supported.

Changed in version 3.10: Improved the algorithm's accuracy so that the maximum error is under 1 ulp (unit in the last place). More typically, the result is almost always correctly rounded to within 1/2 ulp.

`math.`**`sin`**`(x)`

Return the sine of *x* radians.

`math.`**`tan`**`(x)`

Return the tangent of *x* radians.

Angular conversion

`math.`**`degrees`**`(x)`

Convert angle *x* from radians to degrees.

`math.`**`radians`**`(x)`

Convert angle *x* from degrees to radians.

Hyperbolic functions

Hyperbolic functions are analogs of trigonometric functions that are based on hyperbolas instead of circles.

`math.`**`acosh`**`(x)`

Return the inverse hyperbolic cosine of *x*.

`math.`**`asinh`**`(x)`

Module functions example: math

Australian National University

- The functions available in modules would be rather annoying to have to write from scratch each time: `math.log10()`, `math.sin()`, etc
- To use a particular module, one must first import it:

```
>>> import math
```

- Then to use the functions, one must use dot notation:

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)

>>> radians = 0.7
>>> height = math.sin(radians)
```

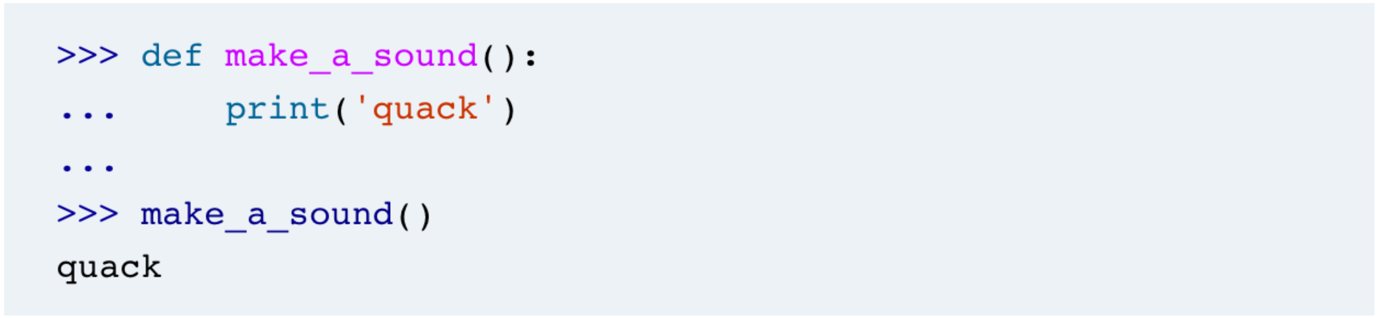
Downey (2015) *Think Python*, 2nd Ed.

Writing your own functions

- Why? If there are parts of your code that you use over and over in a single program, it makes good sense to turn these into a helper function:
  - Shorter code
  - Your code will be easier to read and understand
  - And – the best reason – you only need to change your code in one place when you modify it! It is annoying and very bug-prone to have to make the same changes is multiple different places in your code
  - Eventually, you will end up with a group of helper functions specific to your own work. And you will end up using these over-and-over.

A simple, custom function

- Function definitions start with:
  - the `def` keyword
  - have a name followed by parentheses `()`
  - and a colon `:`
- First this is the definition line. It is followed by an indented code block that does the work of the function:

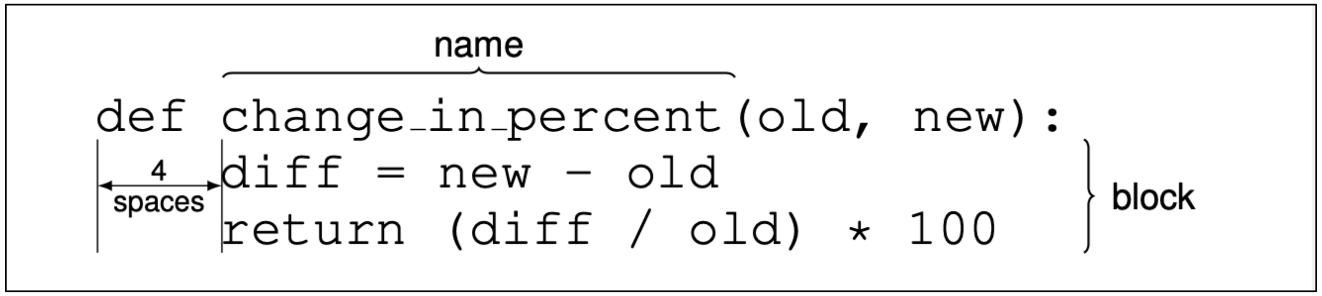


```
>>> def make_a_sound():
...     print('quack')
...
>>> make_a_sound()
quack
```

Lubanovic (2019) *Introducing Python*

- Functions are called by their name, with parentheses `()`
- A function must always be defined before it is called

Function definition



```
def change_in_percent(old, new):
    diff = new - old
    return (diff / old) * 100
```

- A function definition consists of a name and a body (a block)
- The extent of the block is defined by indentation, which must be the same for all statements of a block
  - Standard indentation is 4 spaces
- This example has **parameters**
  - Parameters are specified in the function call, and are passed to the code block
- A custom function must be defined before it can be called

Function parameters and `return` value

Australian National University

```
                      parameters
                     ⏞⏞⏞⏞⏞⏞⏞⏞⏞⏞
def change_in_percent(old, new):
    diff = new - old
    return (diff / old) * 100
```

- Function (formal) parameters are (variable) names
  - These variables can be used only in the function body
- Parameter values will be set only when the function is called
- `return` is a statement
  - when executed, it causes the function call to end, and return the value of the expression

A function call – with parameters

- To call a function, write its name followed by its (actual) **arguments** in parentheses:

```
>>> change_in_percent(489, 556)
13.701431492842536
```

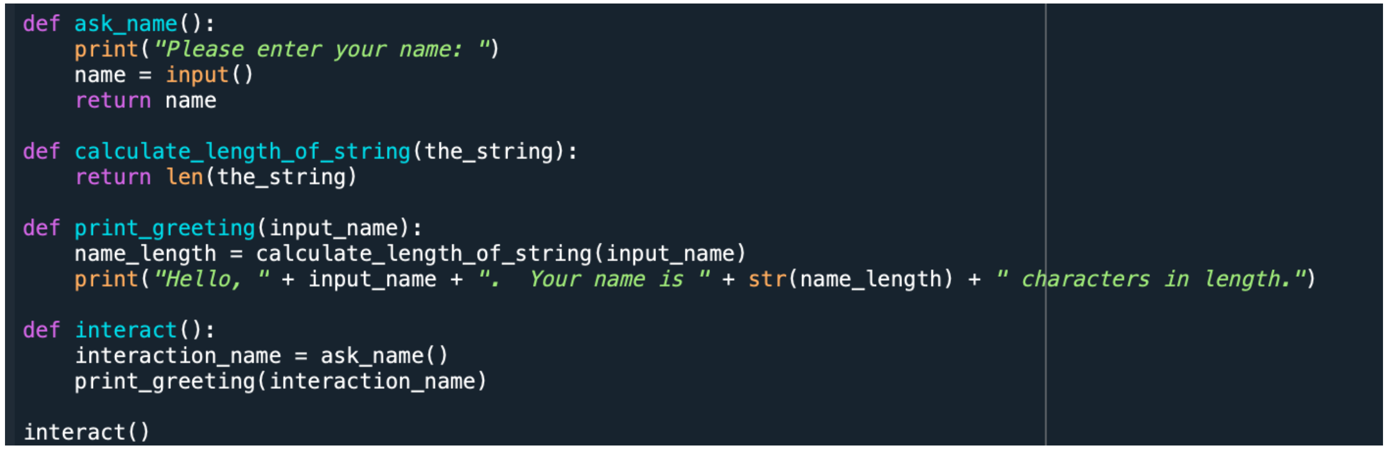
- The arguments are expressions
- Their number should match the parameters
  - Though there can be exceptions – more about this later
- A function call is an expression
  - The call in the example above is an expression that evaluates to the value `return`'d by the function

Terminology: arguments and parameters

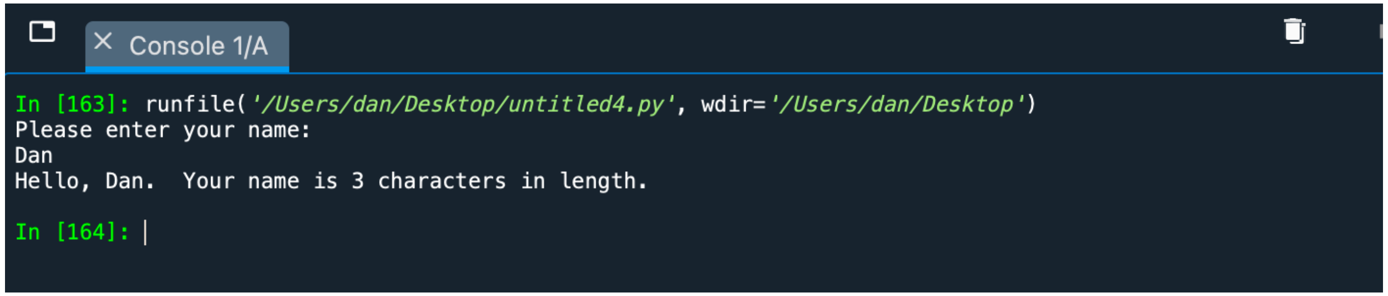
- Arguments are values that are passed to a function when it is called
- Say we make this function call:
  - `print(”ATGTAATAG”)`
  - `print()` is the function
  - `“ATGTAATAG”` is the string argument passed to `print()`
- Parameters are what arguments become when inside the code block within the function

Functions can call other functions

- This is what real-world code is doing all the time. Most code you will write will use other functions to get things done



```python
def ask_name():
    print("Please enter your name: ")
    name = input()
    return name

def calculate_length_of_string(the_string):
    return len(the_string)

def print_greeting(input_name):
    name_length = calculate_length_of_string(input_name)
    print("Hello, " + input_name + ".  Your name is " + str(name_length) + " characters in length.")

def interact():
    interaction_name = ask_name()
    print_greeting(interaction_name)

interact()
```

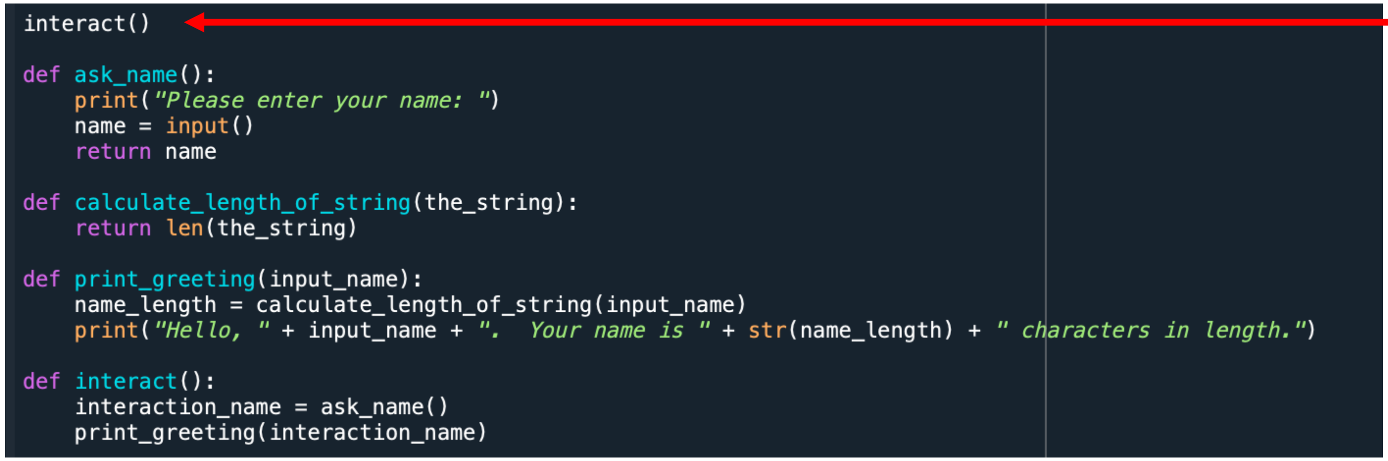


```
In [163]: runfile('/Users/dan/Desktop/untitled4.py', wdir='/Users/dan/Desktop')
Please enter your name:
Dan
Hello, Dan.  Your name is 3 characters in length.

In [164]: 
```

Function definition order

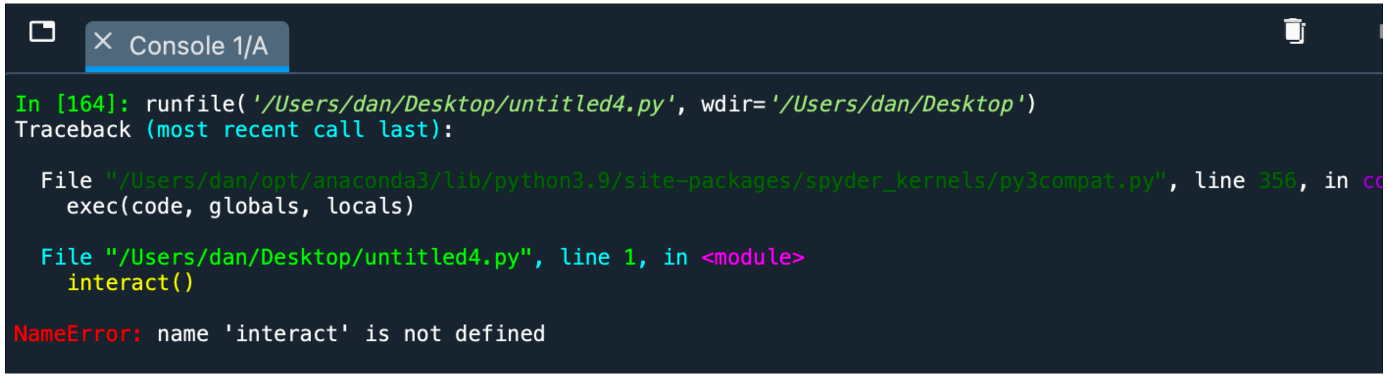
Australian National University

- A function must be defined before it is first called.
- Not like:



```python
interact()

def ask_name():
    print("Please enter your name: ")
    name = input()
    return name

def calculate_length_of_string(the_string):
    return len(the_string)

def print_greeting(input_name):
    name_length = calculate_length_of_string(input_name)
    print("Hello, " + input_name + ".  Your name is " + str(name_length) + " characters in length.")

def interact():
    interaction_name = ask_name()
    print_greeting(interaction_name)
```

Moved function call to program beginning



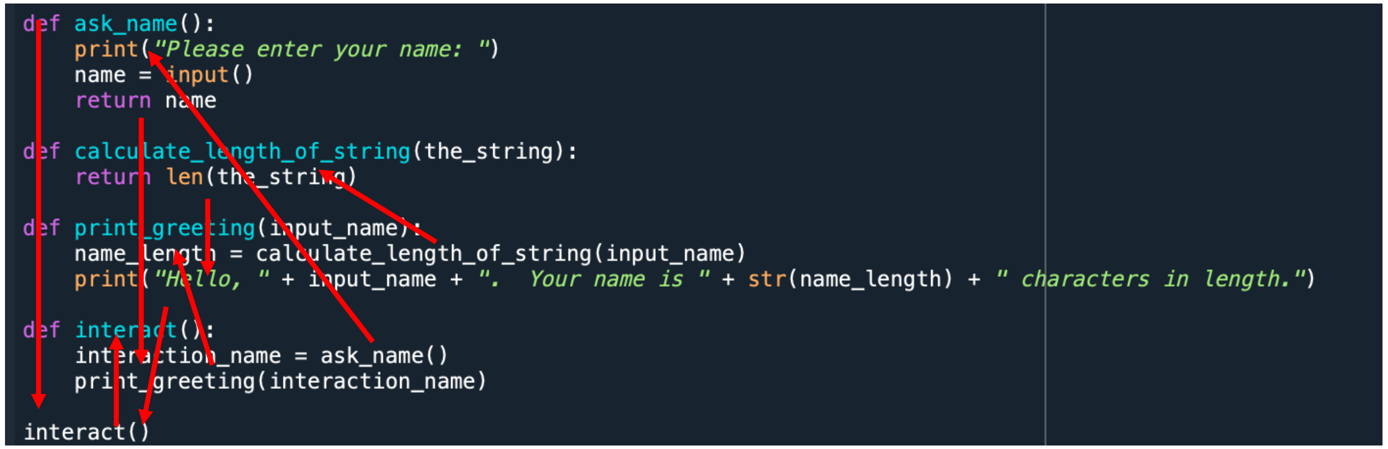
```
In [164]: runfile('/Users/dan/Desktop/untitled4.py', wdir='/Users/dan/Desktop')
Traceback (most recent call last):

  File "/Users/dan/opt/anaconda3/lib/python3.9/site-packages/spyder_kernels/py3compat.py", line 356, in co
    exec(code, globals, locals)

  File "/Users/dan/Desktop/untitled4.py", line 1, in <module>
    interact()

NameError: name 'interact' is not defined
```

Order of evaluation

- The python interpreter always executes instructions one at a time in sequence; this includes expression evaluation
- To evaluate a function call, the interpreter:
  - First, evaluates the argument expressions one at a time, from left to right
  - Then, executes the function body with its parameters assigned the values returned by the arguments expressions
- Same with operators: first arguments (left to right), then the operation

Flow of execution

- Calling a function will interrupt the processive flow of program execution
- Calling a function causes the execution to skip to that function and continue executing from that position

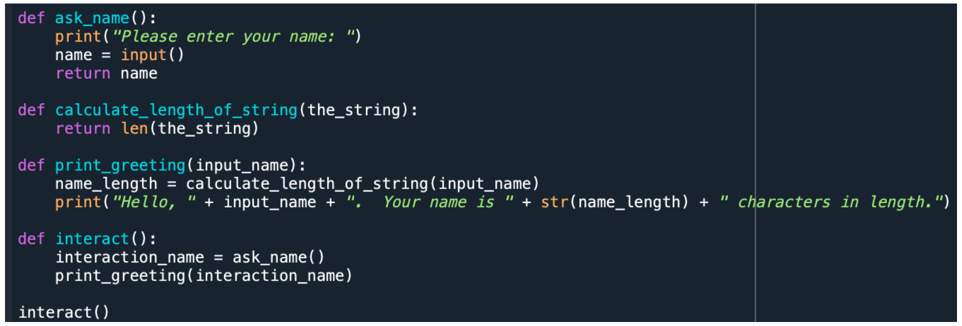


```python
def ask_name():
    print("Please enter your name: ")
    name = input()
    return name

def calculate_length_of_string(the_string):
    return len(the_string)

def print_greeting(input_name):
    name_length = calculate_length_of_string(input_name)
    print("Hello, " + input_name + ".  Your name is " + str(name_length) + " characters in length.")

def interact():
    interaction_name = ask_name()
    print_greeting(interaction_name)

interact()
```

- Execution continues until the end of the function is reached (and it returns to executing where the call was originally made)

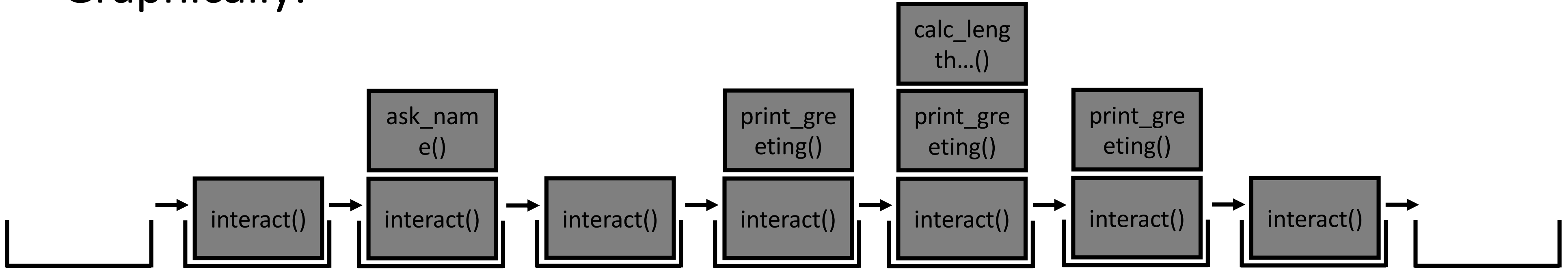
Concept: the call stack

- The ‘to-do list’ of where to come back to after each current function call is called the (execution or call) stack
- When evaluation of a function call begins, the current instruction sequence is put ‘on hold’ while the expression is evaluated – and the function calls begin to ‘stack up’



```python
def ask_name():
    print("Please enter your name: ")
    name = input()
    return name

def calculate_length_of_string(the_string):
    return len(the_string)

def print_greeting(input_name):
    name_length = calculate_length_of_string(input_name)
    print("Hello, " + input_name + ".  Your name is " + str(name_length) + " characters in length.")

def interact():
    interaction_name = ask_name()
    print_greeting(interaction_name)

interact()
```

- Graphically:



Reading: the call stack

- Covered in *Think Python* and Intro to Sci Prog with Python, but..
- A better introduction is in: *Automate the Boring Stuff with Python*, Chapter 3, Section ‘The Call Stack’
- Remember: you have access to the Safari/O’Reilly bookstore through the ANU library: https://www.oreilly.com/library-access/
  - Search for ‘Automate the Boring Stuff with Python’

Example:

- The call stack
- Print statement debugger

`None` values

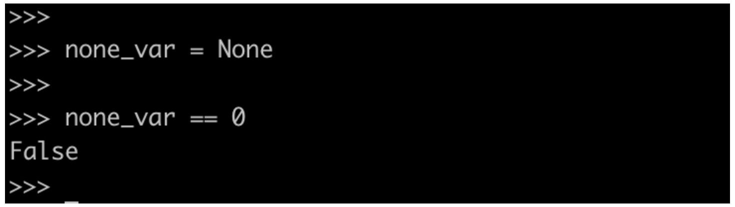
Australian National University

- Some variables contain nothing. Not zero. `None` means null, nothing, undefined.
- `None` type:

```
>>> print(type(None))
<class 'NoneType'>
```

Downey (2015) Think Python, 2nd Ed., Ch 3

- Not the same as zero:



```
>>>
>>> none_var = None
>>>
>>> none_var == 0
False
>>> _
```

- A void value. Just not defined. Some other languages have `NULL` values.
- Why are `NoneType` values useful?

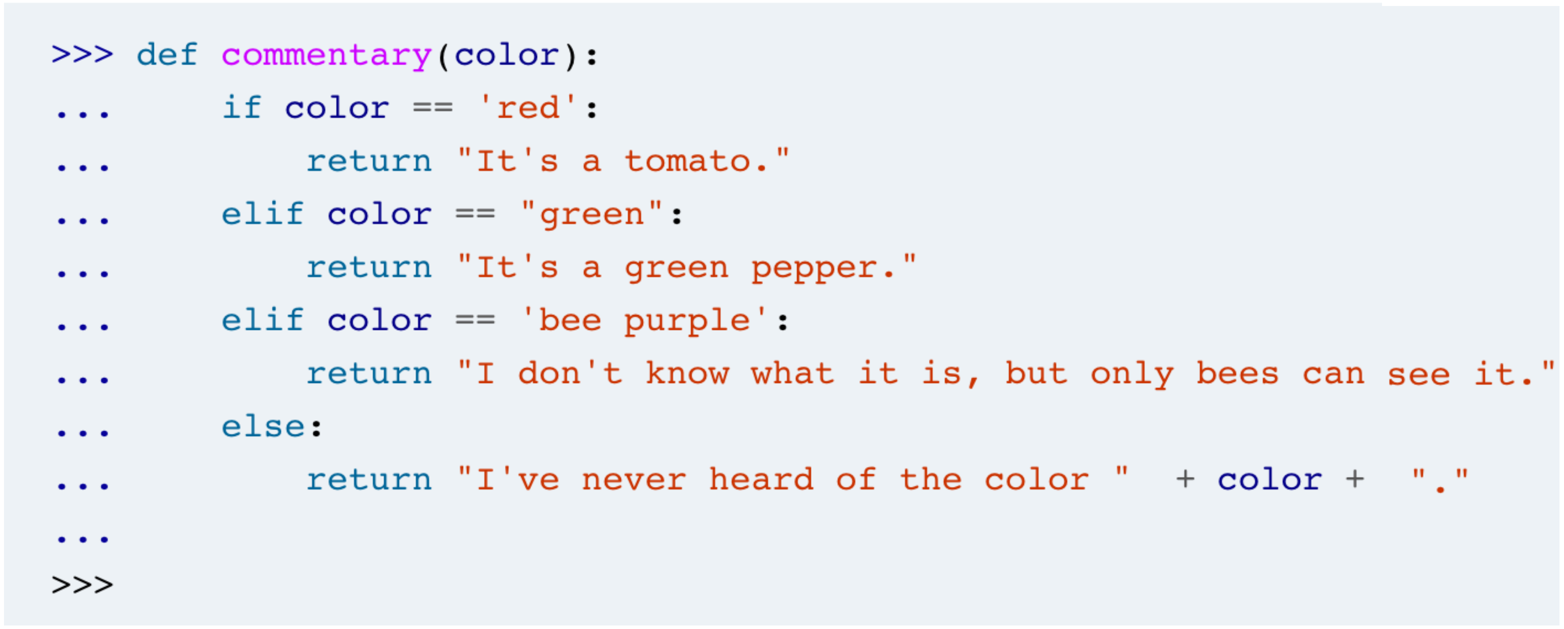
Functions withOUT `return` values

- One place you might encounter `None` is when a function has no `return` statement
- If execution of a function reaches the end of the body without encountering a return statement, the function call returns `None`

- *Note*: with iPython, or interactive mode with Spyder, the interpreter does not print the return value of an expression when the value is `None`.

Multiple `return` statements

- The `return` statement causes execution to leave the function block and return to where a function call was made
- There can be multiple `return` statements in a single function



```
>>> def commentary(color):
...     if color == 'red':
...         return "It's a tomato."
...     elif color == "green":
...         return "It's a green pepper."
...     elif color == 'bee purple':
...         return "I don't know what it is, but only bees can see it."
...     else:
...         return "I've never heard of the color "  + color +  "."
...
>>>
```

Lubanovic (2019) *Introducing Python*

The function docstring

- It is good practice to document your function with a *docstring*
- As simple as a sentence bound with `'''`

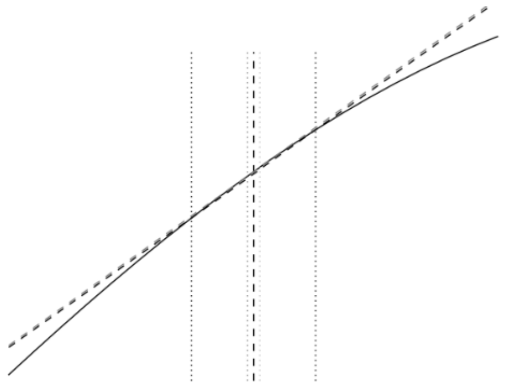
```
def change_in_percent(old, new):
    '''Return change from old to new, as a percentage of the old value.
    Old value must be non-zero.'''
    return ((new - old) / old) * 100
```

- A docstring is a string literal written as the first statement inside a function's body
- Acts like a comment, but accessible through the built-in help system
- Describe *what* the function does (if not obvious from its name) – and its *limits* and *assumptions*

Function calls are *expressions*

- In python, functions are also values: a function can be passed as argument to another function.
- Example: We have a function to compute an approximation of the derivative of a function at a point
- In calling this function, we can provide another function as an argument:

```
def derivative(f, x, d):
    return (f(x + d) - f(x - d)) / (2*d)

ans = derivative(math.sin, math.pi/4, 0.1)
```



Function debugging/testing

- A function is a unit of code:
  - It encapsulates a task, requires specified input and provides as specified output
  - Allows a level of abstraction, wrapping complexity and allows conceptualization
- A function has an interface:
  - It requires certain input – a good function will throw an error if these are not satisfied
  - In turn, it will provide a promised output
  - This is easily tested

Function testing

Australian National University

- A function makes a logical unit for testing:
  - Documented input requirements
  - Expected output
- Testing can run a large variety of cases to ensure correct input produces expected output
- With lots of testing will identify edge-cases - try a range of typical input arguments:
  - values equal to/less than/greater than zero
  - very large and small values
  - values of equal and opposite signs

```
>>> change_in_percent(1, 2)
100.0
>>> change_in_percent(2, 1)
-50.0
>>> change_in_percent(1, 1)
0.0
>>> change_in_percent(1, -1)
-200.0
>>> change_in_percent(0, 1)
ZeroDivisionError
```

Suggested Exercises

- Complete Exercises 3-1, 3-2 and 3-3 of *Think Python*.
- And the Practice Project ‘The Collatz Sequence’ in *Automate the Boring Stuff with Python*, at the end of Chapter 3