Python Quick Micro-Reference Sheet

Installation:
Download the relevant installer for your OS from this page: [Python Release Python 3.12.2 | Python.org]()
If you know how to use your OS package manager, you can use that. Do NOT use Anaconda, conda, miniConda, etc.
If you can open a command prompt and type **python** and receive a Python3 prompt, you're good to proceed.
Otherwise, ask your tutor for help ASAP!

Good Tutorial Sites:
[The Python Tutorial — Python 3.12.2 documentation]()
[Python Tutorial (w3schools.com)]()

Basic Concepts

- python IS CaSe SeNsiTiVe
- indentation (whitespace) defines blocks (not braces {…} or begin/end). Use spaces.
- `#` for single line comments (no block comment)
- `=` is for assignment, and assignments creates references - not copies - of objects
- `==` is for logical comparison, also `<= != < > >=`
- Python uses ***reference counting*** and ***garbage collection***
- Logical operators in words `and`, `or` and `not`, with `True` and `False` types
- ; can be used to terminate lines or separate statements (but not recommended)
- : is used to define structure or control (see `if`, `else`, `while` flow control)
- `None` is a special constant (think NULL, void or nil), logical `False`

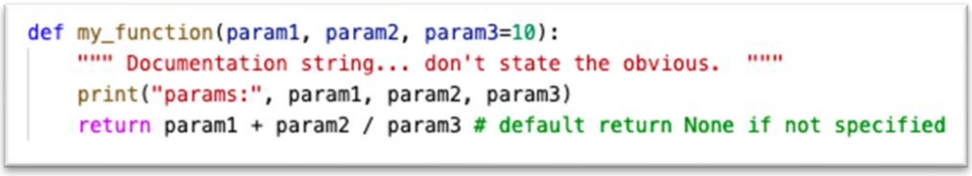
Sequence Types

- index with [index], start at zero, range selection ("slice") with [start:end]
- `tuple1 = (1,'a',None,3.42,'hello')` # tuple - immutable, ordered, mixed types
- `list1 = ['abc', None, 5, 7.4]` # list - mutable, ordered, mixed types
- `str1 = 'have a nice day!'` # string defined with either set of quotation marks '…', "…"
- `c = a + b` # will create new tuple/list/string (lists can have .extend(..) or .append(…))
- `c = a * 3` # will create new multiple of tuple/list/string

Dictionary

Stores a mapping (hash) between key/values. We can define, view, delete etc.
Unordered. Mutable. Mixed types. (Nested.) Keys are unique.
```
dict1 = {'key1': 'yippy', 'key2': 1234, 'key12', None}
```

Procedures/Functions



```
def my_function(param1, param2, param3=10):
    """ Documentation string... don't state the obvious.  """
    print("params:", param1, param2, param3)
    return param1 + param2 / param3 # default return None if not specified
```

No function overloading (but you can specify default parameter values etc)
Functions ***are*** objects and can be passed around just like variables

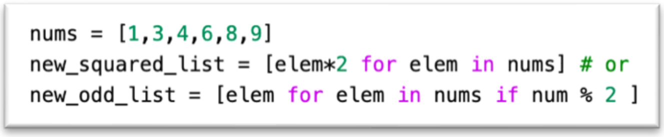
Flow Control

`if test1:`
`    …`
`elif test2:`
`    …`
`else:`
`    …`
	`while test_is_true:`
	`    …`
	`    if move_on_test:`
	`        continue`
	`    if break_test:`
	`        break`
		`for item in collection:`
		`    print(item)`
		# works on any collection or
		# iterable object
		`for i in range(10):` # creates an iterable range
		`    print(i)`

```
assert (current_score < 10), 'Hey, why did that happen? '
x = true_value if condition else false_value
```

List Comprehensions

Easy way to create lists, often using existing list/iterables. It's a very popular python technique.
They look a bit like a `for` loop and an `if` statement (optional). They can be nested (gets messy).



```
nums = [1,3,4,6,8,9]
new_squared_list = [elem*2 for elem in nums] # or
new_odd_list = [elem for elem in nums if num % 2 ]
```