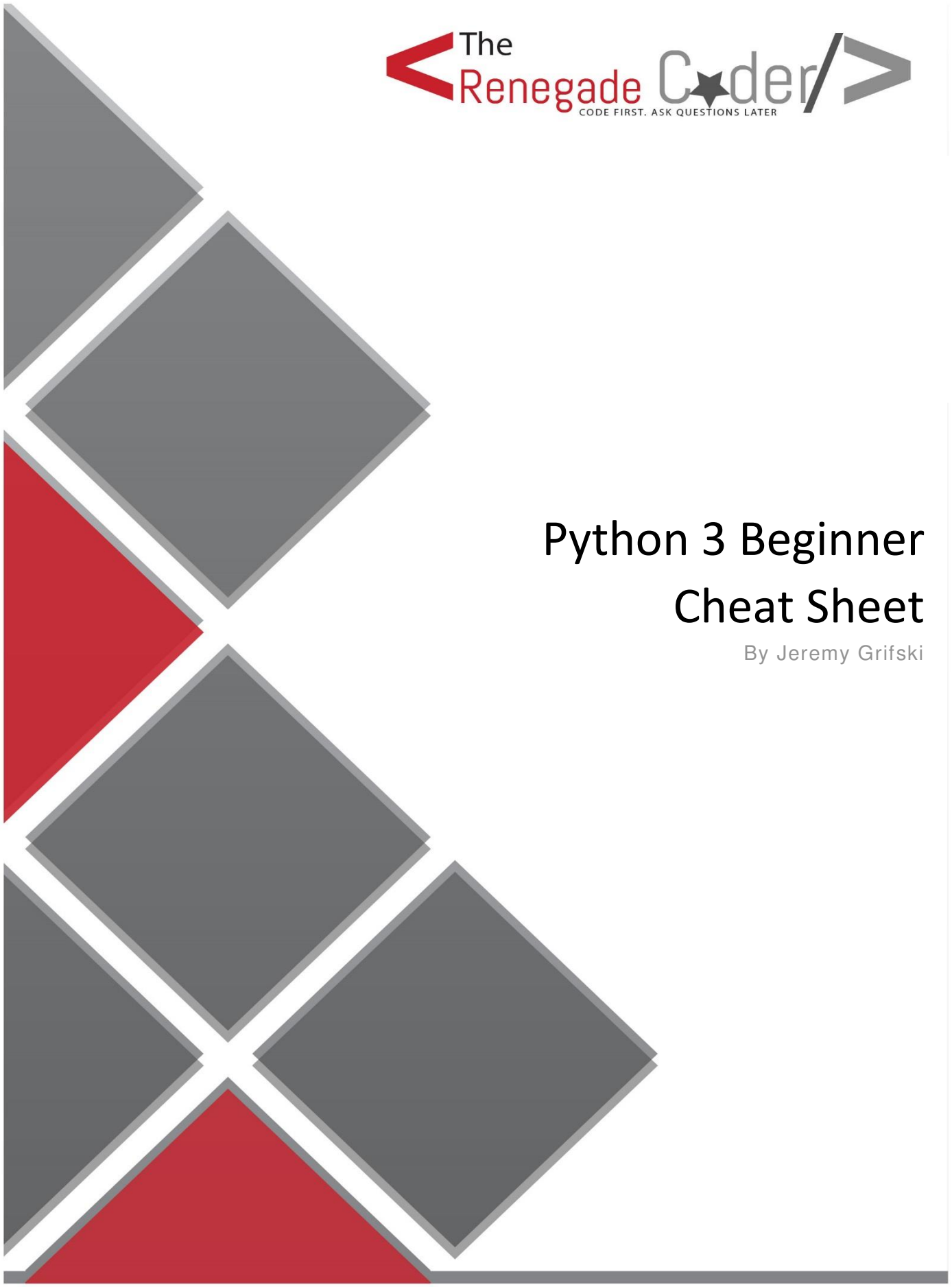

Python 3 Beginner Cheat Sheet

By Jeremy Grifski

Variables

Variables can be created and stored as follows: *name = expression*.

```
num = 11  # stores 11
word = "Hello"  # stores "Hello"
logic = True  # stores True
my_list = [2, 3, 4]  # stores [2, 3, 4]
my_tuple = (5, 2)  # stores (5, 2)
my_dict = {} # stores {}
my_set = {4, 5}  # stores {4, 5}
```

Arithmetic

Arithmetic expressions can be written using these operators: +, -, /, //, *, **, and %.

```
add = 3 + 7  # stores sum of 10
sub = 5 - 3  # stores difference of 2
mul = 3 * 3  # stores product of 9
div = 5 / 2  # stores quotient of 2.5
idiv = 5 // 2  # stores quotient of 2
exp = 2 ** 4  # stores product of 16
rem = 7 % 2  # stores remainder of 1
paren = (3 + 2) / 5  # stores 1.0
```

Boolean Algebra

Boolean algebra is accomplished using the following keywords: *and*, *or*, and *not*.

```
is_cold = True # stores True
is_wet = True # stores True
is_not_cold = not is_cold # stores False
am_sad = is_cold and is_wet # stores True
am_mad = is_cold or is_wet # stores True
```

Relational Operators

Numbers can be compared using the relational operators: >, >=, ==, <=, and <.

```
less = 2 < 4  # stores True
less_equals = 3 <= 3  # stores True
equals = 4 == 7  # stores False
greater_equals = 5 >= 2  # stores True
greater = -4 > 5  # stores False
```

Strings

Strings are collections of characters that can be created using quotation marks.

```
name = "Jeremy"  # stores "Jeremy"
size = len(name)  # stores length of 6
twice = name * 2  # stores "JeremyJeremy"
concat = name + "'s"  # stores "Jeremy's"
check = "e" in name  # stores True
first = name[0]  # stores "J"
last = name[-1]  # stores "y"
subset = name[1:4]  # stores "ere"
lower = name.lower()  # stores "jeremy"
upper = name.upper()  # stores "JEREMY"
```

Dictionaries

Dictionaries are collections of key, value mappings and are created using braces.

```
my_map = {"x": 2} # stores {"x": 2}
value = my_map["x"] # stores 2
my_map["y"] = 5 # adds "y": 5 to dict
list(my_map.keys()) # returns [x, y]
list(my_map.values()) # returns [2, 5]
```

Lists

Lists are mutable collections which can be created using square brackets.

```
items = [1, 2, 3]  # stores [1, 2, 3]
length = len(items)  # stores 3
begin = items[0]  # stores 1
end = items[-1]  # stores 3
section = items[0:1]  # stores [1]
exists = 2 in items  # stores True
items.append(4)  # adds 4 to end of list
items.extend([5, 6])  # appends 5 and 6
items.reverse()  # flips order of list
items.clear()  # empties list
```

Conditionals

In Python, conditions are written using the *if/elif/else* syntax.

```
# prints "Nice car!"
cars = ["Tesla", "Ford", "Toyota"]
if "Toyota" in cars:
    print("Nice car!")
elif "Audi" in cars:
    print("Not bad!")
else:
    print("Mistakes were made.")
```

In addition, if statements can be nested.

```
# removes "Toyota" from cars
# prints "We won't be needing that!"
if "Toyota" in cars:
    if "Tesla" in cars:
        cars.remove("Toyota")
        print("We won't be needing that!"
```

Loops

In order to repeat a section of code, Python includes both *while* and *for* loops.

```
# prints "h\ni\n"
greet = "hi"
index = 0
while index < len(greet):
    print(greet[index])
    index += 1

# same code, but with a for loop
for index in range(len(greet)):
    print(greet[index])

# same code, but with a for each loop
for char in greet:
    print(char)
```

Comprehensions

Finally, lists and dictionaries can be generated using comprehensions.

```
# creates list from 0 to 49
nums = [x for x in range(50)]

# creates list of all nums doubled
doubles = [x * 2 for x in nums]

# creates list of evens from nums
evens = [x for x in nums if x % 2 == 0]

# creates a mapping of nums to evens
map = {x: y for x, y in zip(nums, evens)}
```