

Python 'Cheat Sheet'

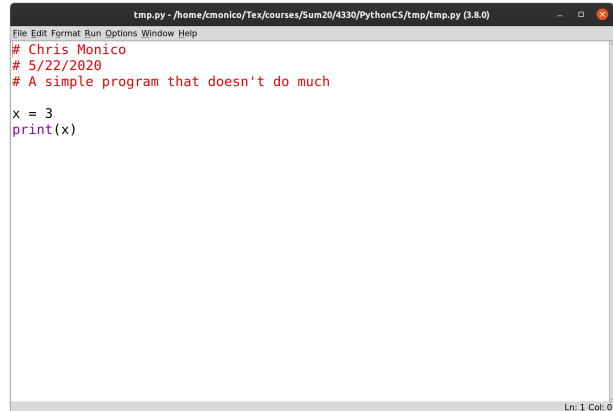
May 28, 2020, Chris Monico

This is just a very quick reference, with a couple of small examples illustrating each concept. Most of these are far more powerful than the examples illustrate, though!



```
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Python 3.8.0 (default, Oct 28 2019, 16:14:01)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /home/cmonico/Tex/courses/Sum20/4330/PythonCS/tmp/tmp.py
3
>>> |
```

IDLE shell window



```
tmp.py - /home/cmonico/Tex/courses/Sum20/4330/PythonCS/tmp/tmp.py (3.8.0)
File Edit Format Run Options Window Help
# Chris Monico
# 5/22/2020
# A simple program that doesn't do much

x = 3
print(x)
```

IDLE editor Window

IDLE

Start a new program	in shell window, <i>File</i> → <i>New File</i> , or CTRL+N
Open existing program	in either window, <i>File</i> → <i>Open File</i> , or CTRL+O
Run program	in editor window, <i>Run</i> → <i>Run Module</i> , or F5

Comments

This comments out a single line

Or surround with triple-quotes, for multi-line comments.

Variables

Variable names are case-sensitive, and can contain upper and lower case letters, digits, and the underscore character. They may **not** start with a digit. The following are valid and different variable names:

```
N = 10 #int
n = 2 #another int
my_str = 'Monty' #string
my_str2 = "Python" #string
x2 = 1.2917 #float
goodjob = True #bool
```

Numeric Operators

For numbers, the basic arithmetic operators are exactly what you would expect: `+` `-` `*` `/`, and parentheses group expressions as you would expect. Three additional operators that are often useful are the modulus operator `%`, integer division operator `//`, and exponentiation operator `**`

```
x = 3
y = 2
# (1) The modulus operator % to compute the remainder of x divided by y:
r = x % y
# The integer division operator to return the integer quotient:
q = x // y
# (2) The exponentiation operator ** to compute x to the y power:
u = x**y
v = (x+y)**(0.5)
```

Printing

```
n=5
pi=3.141592653589
# Simple positional formatting:
# %d integer, %f float, %s string
print("n is %d and pi is about %1.5f" % (n,pi))
# The format method:
print("n is {0} and pi is about {1}".format(n,pi))
```

Input

```
# Prompt the user to enter a name
name = input("Enter a name: ")
# Prompt the user for an age, but convert to an int,
# in case we want to do arithmetic with it later.
age = int(input("Age: "))
print("Name: %s, Age: %d" % (name, age))
```

for loops

```
s=0
for n in range(4):
    s = s+n
print(s) #prints 6, since 0+1+2+3=6.
The above code is the same as:
print(sum(range(4)))
which is also the same as: print(sum([0,1,2,3]))
```

while loops

Suppose we want to find the least positive odd integer N for which $N^3 + 3N^2 > 1000$. We can check 1, 3, 5,... in order until we find one that works. A `for` loop is not an ideal choice, because we don't know exactly how many times we need to iterate.

#Find the smallest odd positive integer N for which

$N^3 + 3N^2 > 1000$,

N=1

#Note: this loop will terminate, because we know such an N exists.

```
while N**3 + 3*N**2 <= 1000:
```

```
    N += 2
```

```
print(N)
```

Conditional statements

```
x = float(input("Enter a number: "))
```

```
if x>0:
```

```
    print("your number is positive")
```

```
elif x<0:
```

```
    print("your number is negative")
```

```
else:
```

```
    print("your number is zero.")
```

Note: since `=` is the assignment operator, there is a different operator for testing equality:

```
if x == 0:
```

```
    print("x is zero.")
```

Functions

```
def sumofdigits(n):  
    # Given a positive integer n,  
    # return the sum of its digits .  
    s = 0  
    remaining = n  
    while remaining > 0:  
        s += (remaining%10) #Add the last digit to s  
        remaining = remaining // 10 #Remove the last digit  
    return s  
  
k = 3915  
# The function sumofdigits will be called with the argument 3915,  
# and the value it returns will be substituted in place:  
res = sumofdigits(k)  
print("sum of the digits of {0} is {1}.".format(k, res))  
# try to print(remaining) here and see what happens.  
# The variable no longer exists , so it would be an error.  
# This is an example of 'scope'.
```

Lists

```
my_list=[3,1,4,1,5,9,2,6]  
print(my_list[0]) #prints the number 3  
print(len(my_list)) #prints 8.  
#range creates a list of integers in a given range.  
print(range(4)) #prints [0,1,2,3]  
print(range(1,4)) #prints [1,2,3]  
  
List comprehension is like mathematical set-builder notation. Compare the following with  
the set  $\{x^2 : x \in \mathbb{Z} \cap [0, 10)\}$ :  
  
squares = [x**2 for x in range(10)]  
10 in squares #evaluates to False  
16 in squares #evaluates to True  
  
Other useful list methods include insert, append, index, pop.
```