Python basics reference for beginners

This summary is not intended as a teaching guide but a reminder on how the basic features of Python work. It is intended for use after having done an introduction to Python.

My detailed Python notes can be found at https://pbaumgarten.com/python

Last updated: January 2020 by P.Baumgarten

Print and input

To print text to screen:

```
print("Hello world!")
```

To print a variable to screen:

```
name = "Han Solo"
record = 12
print(f"{name} completed the Kessel run in {record} parsecs")
# notice the 'f' in front of the first set of quotes
```

To ask the user for input

```
name = input("What is your name? ")                          ## Input saved as text string
num = int(input("Type an integer between 1 and 100: "))      ## Input saved as an integer
double = num * 2
print(f"Hello {name}, double your number is {double}")
```

Sometimes you need to format the variables you are printing.

```
val = 12
print( f"With leading spaces to make it 4 characters wide is {val:4}" ) # prints '  12'
print( f"With leading zeros to make it 4 characters wide is {val:04}" ) # prints '0012'
val = 3.14
print( f"To 2 decimal places is {val:.2f}" ) # prints '3.14'
print( f"To 5 decimal places is {val:.5f}" ) # prints '3.14000'
print( f"With spaces to make it 8 characters wide with 3 decimal places is {val:8.3f}" )
print( f"With zeros to make it 8 characters wide with 3 decimal places is {val:08.3f}" )
```

Numbers

Python has two types of numbers: integers and floats. Integers are "whole numbers" without decimals, floats are the name given to numbers that contain decimals.

To get the result of a mathematical calculation, put the equation on the right of an equal sign, and the variable you wish the answer saved in on the left of the equal sign.

Arithmetic

```
a = 100
b = 6
c = a + b           # addition          ... c == 106
c = a - b           # subtraction       ... c == 94
c = a * b           # multiplication    ... c == 400
c = a / b           # division          ... c == 16.66667
c = a // b          # integer division  ... c == 16 (how many times does 6 go into 100)
c = a % b           # modulus remainder ... c == 4 (ie: remainder of 100 divided by 6)
c = a ** b          # exponent          ... c == 1000000000000 (ie: 10^6)
```

Geometry and trigonometry

```
import math         # add this line to the top of your program for the math functions to work
```

- `math.pi` - returns value of pi with as much precision as available to your computer
- `math.hypot( a, b )` - returns the hypotenuse for a right angled triangle with side lengths a and b
- `math.sin( radians )` - returns the sin() for an angle provided in radians
- `math.cos( radians )` - returns the cos() for an angle provided in radians
- `math.tan( radians )` - returns the tan() for an angle provided in radians
- `math.asin( ratio )` - returns the inverse sin() for a ratio. answer provided in radians
- `math.acos( ratio )` - returns the inverse cos() for a ratio. answer provided in radians
- `math.atan( ratio )` - returns the inverse tan() for a ratio. answer provided in radians
- `math.degrees( radians )` - convert angle from radians to degrees
- `math.radians( degrees )` - convert angle from degrees to radians

Example: How long is the hypotenuse of a triangle if the adjacent side is 20, and the angle is 45 degrees?

```
import math
adjacent = 20.0
angle = 45.0
hypotenuse = adjacent / math.cos( math.radians( angle ))
print(hypotenuse)       # prints 28.284...
```

Converting

```
new_integer = int( "10" )   # convert string "10" to integer 10
new_float = float( "3.14" ) # convert string "3.14" to float 3.14
new_string = str( 42 )      # convert integer 42 to string "42"
```

Strings

Assign a text string a value

```
mytext = "Hello"
```

Searching strings

```
mytext = "To infinity and beyond!"
if "infinity" in mytext:
   print("Yes, the word infinity is in the string")
else:
   print("No, the word infinity is not in the string")
```

Get substrings

- String positions start from 0. That is, the first letter is position 0, the second letter is position 1 and so forth.
- When asking for a range of characters, Python will give you a substring that includes the starting position number, but not including the end position number.

```
original_text = "To infinity and beyond!"
new_text = original_text[:2]    ## Get from start up to not including position 2, ie: "To"
new_text = original_text[16:]   ## Get from position 16 to end, ie: "beyond!"
new_text = original_text[3:11]  ## Get from position 3 up to not position 11. ie: "infinity"
```

Changing strings

```
original_text = "To infinity and beyond!"
new_text = original_text.lower()              ## == "to infinity and beyond!"
new_text = original_text.upper()              ## == "TO INFINITY AND BEYOND!"
new_text = original_text.title()              ## == "To Infinity And Beyond!"
new_text = original_text.swapcase()           ## == "tO INFINITY AND BEYOND!"
new_text = original_text.ljust(30)            ## == "To infinity and beyond!       "
new_text = original_text.rjust(30)            ## == "       To infinity and beyond!"
new_text = original_text.replace(" ", "--")   ## == "To--infinity--and--beyond!"
```

Query content of string

```
text = "To infinity and beyond!"
num = len(text)             ## get length of string ... num == 23
num = text.count(" ")       ## count spaces in string ... num == 3
num = text.index("o")       ## position of first 'o' in the string ... num == 1
num = text.rindex("o")      ## position of last 'o' in the string ... num == 19
result = text.isnumeric()   ## does it contain only numbers?
result = text.isalpha()     ## does it contain only letters?
result = text.islower()     ## is it all lowercase?
result = text.isupper()     ## is it all uppercase?
result = text.istitle()     ## is it all title case?
result = text.isspace()     ## is it all spaces?
```

If statements

An "if" statement defines code that will run if the answer to a question is `True`. To ask a question, have Python to compare two or more values to see if they obey a rule. Examples of comparisions we can ask...

```
if 1 == 1:              ## Is 1 equal to 1              ... True
if 1 == 0:              ## Is 1 equal to 0              ... False
if "a" == "a":          ## Is "a" equal to "a"          ... True
if "a" == "A":          ## Is "a" equal to "A"          ... False
if "a" != "z":          ## Is "a" not equal to "z"      ... True
if 1 > 0:               ## Is 1 greater than 0          ... True
if -1 > 0:              ## Is -1 greater than 0         ... False
if 2 >= 3:              ## Is 2 greater or equal to 3   ... False
if -3 < -1:             ## Is -3 less than -1           ... True
if 3 < 1:               ## Is 3 less than 1             ... False
if 2 <= 3:              ## Is 2 less or equal to 3      ... True
```

We can also query string content such as:

```
text = "May the force be with you!"
if "force" in text:
    print("The force is strong with this string")
else:
    print("The force is not with this string")
```

We can also join multiple queries together using the `and` or `or` key words such as...

```
a = int(input("Enter a number: "))
if a > 0 and a < 10:
    print("You entered a number between 0 and 10")
if a < 0 or a > 10:
    print("You entered a number less than 0 or greater than 10")
```

You can join multiple `if` queries together using `elif`.

```
a = int(input("Enter a number: "))
if a > 10:
    print("a is bigger than 10")
elif a > 0:
    print("a is bigger than 0 but not bigger than 10")
elif a == 0:
    print("a is zero")
else:
    print("a is less than 0")
```

Remember:

- Use a double equal sign to compare two values! A single equal is used to **set** the value rather than ask if they are a match.
- End your "question" with a colon and indent the code to run when the comparison is True.
- The `if` statement will keep asking questions of the various `elif` until it finds one that is `True`. After one item is `True`, it will skip the rest of the options available.

While loops

The `while` loop works very similar to the if statement. Any question you can ask of an `if` statement can be used in a `while` loop. The difference being that so long as something is `True`, it will keep running the same indented section of code. An example:

```
stop_at = int(input("Enter a number for me to count up to: "))
num = 1
while num <= stop_at:
   print( num )
   num = num + 1
print("The end!")
```

Another example...

```
import random
secret = random.randint(1,99)   # generate a random number between 1 and 99
guess = int(input("Guess my secret number between 1 and 99: "))
while guess != secret:
    if guess > secret:
        guess = int(input("Too high. Guess again: "))
    elif guess < secret:
        guess = int(input("Too low. Guess again: "))
print("Correct!")
```

For loops

You can also use a for-loop when you know the number of iterations you wish to loop in advance.

```
limit = int(input("Enter a number for me to count up to: "))
for number in range(limit):   # will loop from 0 to limit-1
   print( number )
print("The end!")
```

You can also specify a starting number other than zero. For instance

```
for number in range(50, 100):   # will loop from 50 to 99
   print( number )
print("The end!")
```

You can even specify that it counts downwards, or using an interval different to one by specifying a third parameter to the `range()` function.

```
for number in range(100, 0, -1):   # will loop from 100 to 1
   print( number )
print("The end!")
```

Lists and for-loops

A list is a means of storing multiple values to one variable name. Other programming languages call these 'arrays'.

Example lists:

```
primes = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
vowels = ["A", "E", "I", "O", "U"]
starwars = ["Luke", "Han", "Leah", "Obi-wan", "Yoda", "Rey", "Finn"]
```

Lists have many of the same features of strings (which is really just a list of characters) to query them and get sub-parts from.

```
size = len( starwars )              ## How many items in the starwars list
first = starwars[0]                 ## Get the first item in the starwars list
second = starwars[1]                ## Get the second item in the starwars list
last = starwars[-1]                 ## Get the last item in the starwars list
starwars.append("Darth Vadar")      ## Add an item to the starwars list
starwars.sort()                     ## Sort the list into alphabetical or numerical order
smallest = min(primes)              ## Get the smallest value from the list
largest = max(primes)               ## Get the largest value from the list
```

You can query if an item exists inside a list

```
starwars = ["Luke", "Han", "Leah", "Obi-wan", "Yoda", "Rey", "Finn"]
if "Luke" in starwars:                  ## Is "Luke" in the starwars list?
    print("Luke is in starwars")
else:
    print("Luke is not in starwars")
```

We can also use a `for` loop to process every item in a list

```
starwars = ["Luke", "Han", "Leah", "Obi-wan", "Yoda", "Rey", "Finn"]
for character in starwars:
    print(f"{character} is a person in Starwars")
```

Defining functions

A function allows us to define a block of code as our own Python command to use. One useful purpose of this is it allows you to reuse code without having to continually copy-and-paste it. This then means you can also modify/improve it by only editing it once. Very useful.

Use the `def` keyword to define a new function, provide the name you wish to assign, and then parenthesis and a colon. Indent the code to include in the function and then end the intendation when the function specific code ends.

```
# Create a function...
def a_useless_function():
    print("This is a fairly useless function")

# Execute the function 3 times...
a_useless_function()
a_useless_function()
a_useless_function()
```

You can provide parameters to functions so their behaviour can be customised each time. A simple example...

```
def area_of_triangle( base, height ):
    area = 0.5 * base * height
    return area # send this value back to the code that ran the function

print( area_of_triangle( 10.0, 15.0 ))  # prints 75.0
print( area_of_triangle( 7.0, 12.0 ))   # prints 42.0
print( area_of_triangle( 4.0, 5.5 ))    # prints 11.0
```

Files

Read entire file as a string

```
with open("countries.txt", "r") as f:   # Open file for reading
    content = f.read()                  # Load entire file into 1 large string
    print(content)                      # Do something with the string
```

Read entire file as a list, one string per line

```
with open("countries.txt", "r") as f:   # Open file for reading
    content = f.read()                  # Load entire file into 1 large string
    lines = content.splitlines()        # Split into lines
    for line in lines:
        print(line)
```

Writing a text file - Using just a string

```
content = "My exciting material"
with open("stuff.txt", "w") as f:   # Open file stuff.txt for writing
    f.write(content)                # Write this string to the file
```

Writing a text file - Using a list of strings

```
content = ['Leah', 'Obi-wan', 'Yoda', 'Rey', 'Finn', 'bb-8']
save = "\n".join(content)           # Convert list to a string, adding a new-line character
after each string
with open("people.txt", "w") as f:  # Open file people.txt for writing
    f.write(save)                   # Write this string to the file
```

Add to a file without replacing the original content

```
content = "More exciting material"
with open("stuff.txt", "a") as f:   # Open file stuff.txt for appending
    f.write(content)                # Add this string to the file
```

File opening modes:

- `r` for reading
- `w` for writing (erasing it if it exist)
- `a` for appending (add to file without erasing previous content)

Remember

- The `with` statement will close the file when you unindent
- `.splitlines()` behaves like `.split("\n")`
- For greater predictability, cast everything to strings before writing to files

Exceptions

Generic try/except

- Warning the generic exception catch is bad practice and hides bugs. Any unintentional error in your code (such as a mis-spelling) could cause an exception that results in hours of frustration to diagnose.

```
try:
    denominator = int(input("Please enter a number: "))
    result = 100 / denominator
    print(f"100 divided by {denominator} is {result}")
except:
    print("I can't do that!")
```

Try/except checking for specific error types

```
try:
    denominator = int(input("Please enter a number: "))
    result = 100 / denominator
    print(f"100 divided by {denominator} is {result}")
except ValueError:
    print("That wasn't a number")
except ZeroDivisionError:
    print("I can't divide by zero")
```

Generate your own exception

```
x = 10
if x > 5:
    raise ValueError("Not allowed to have a number greater than 5")
```

Dates and times

Creating a datetime

```
from datetime import datetime

# Create a datetime using current computer date & time
now = datetime.now()

# Create a datetime with year=2019, month=12, day=25
christmas = datetime( 2019, 12, 25 )

# Create a datetime with year=2019, month=12, day=25, hour=11, minute=00, seconds=00
christmas = datetime( 2019, 12, 25, 11, 00, 00 )

# Create a datetime from a formatted string
birth_text = input("What is your birthday (write it as dd/mm/yyyy) ?")
birth_date = datetime.strptime( birth_text, "%d/%m/%Y" )
```

Using timestamps (number of seconds since 01/01/1970 00:00 UTC)

```
from datetime import datetime

# Create a timestamp based on current date/time
timestamp = datetime.now().timestamp()

# Create a timestamp from existing date/time object
apollo_11 = datetime( 1969, 7, 20, 20, 17, 40 )
timestamp = apollo_11.timestamp()       # -14215340.0

# Create a datetime from a timestamp
timestamp = 1563958625      # Number of seconds since 01/01/1970 00:00 UTC
july24_2019 = datetime.fromtimestamp(timestamp)
```

Differences between dates with `timedelta`

- `timedelta` acceptsany combination of the following options: days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0

```
from datetime import datetime, timedelta

apollo_11 = datetime( 1969, 7, 20, 20, 17, 40 )
now = datetime.now()

# Create a timedelta automatically by subtracting two dates
diff = now - apollo_11
print(f"Apollo 11 landed {diff.days} days ago!")

# Create a new date by adding a timedelta to a date
ten_thousand = apollo_11 + timedelta( days=10000 )
print(f"10'000 days after Apollo 11 was { tenthousand.strftime("%d %B, %Y") }")
```

Create pretty date/time strings using `strftime()`

```
pretty_date_1 = apollo_11.strftime("%A, %d %B, %Y")  # 'Sunday, 20 July, 1969'
pretty_date_2 = apollo_11.strftime("%d/%m/%Y")       # '20/07/1969'
pretty_time = apollo_11.strftime("%H:%M:%S")         # '20:17:40'
```

Date based codes

- %a - Weekday abbreviated (eg: Sun)
- %A - Weekday full name (eg: Sunday)
- %d - Day number in month (zero padded eg: 02)
- %b - Month name abbreviated (eg: Jan)
- %B - Month full name (eg: January)
- %m - Month number (zero padded eg: 01)
- %y - Year without century (zero padded)
- %Y - Year with century (zero padded)

Time based codes

- %I - Hour 12 hour clock (zero padded)
- %H - Hour 24 hour clock (zero padded)
- %M - Minute (zero padded)
- %S - Second (zero padded)
- %p - AM or PM

Get parts of date/time

```
from datetime import datetime
apollo_11 = datetime( 1969, 7, 20, 20, 17, 40 )
y = apollo_11.year        # 1969
m = apollo_11.month       # 7 (July)
d = apollo_11.day         # 20
hr = apollo_11.hour       # 20 (8:00pm in 24 hr time)
mi = apollo_11.minute     # 17
se = apollo_11.second     # 40
wkd = apollo_11.weekday() # 6 (0=Monday so 6 is Sunday)
```

Replace parts of a date using `replace()`

```
from datetime import datetime
date_1 = datetime(1980, 6, 20)
date_2 = date_1.replace( year = 2019 )  # Replace the year
print(date_2)   # 20/06/2019
```

Dictionaries

```
# Create an empty dictionary
person = { }    # Curly braces instead of the square brackets used for lists

# Set values to your dictionary
person["given_name"] = "Paul"
person["family_name"] = "Baumgarten"

# Get elements from the dictionary
print( person["given_ame"] )
print( person["family_name"] )

# Add / modify elements in the dictionary
person["email"] = "pbaumgarten@isl.ch"
person["website"] = "https://pbaumgarten.com"

# Remove an element from the dictionary
del person["website"]
```

Loop through all the elements of the dictionary

```
for key,val in person.items():
   print(f"field {key} has value {val}")
```

Convert a dictionary/list structure into a JSON string (useful for saving to a file)

```
import json

# Convert to JSON text string
json_text = json.dumps( person )

# Save it to a file
with open("person.txt", "w") as f:
    f.write( json_text )
```

Convert a JSON string into a dictionary/list structure (usefil for loading from a file)

```
import json

# Load from a file
with open("person.txt", "r") as f:
    content = f.read()

# Convert string text to dictionary/list structure
person = json.loads( content )
```