**Python can be used on a server to create web applications.**

Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Indentation
Indentation refers to the spaces at the beginning of a code line.
Python uses indentation to indicate a block of code.

Variables are containers for storing data values.

Casting
If you want to specify the data type of a variable, this can be done with casting.

| | |
|---|---|
| Text Type: | `str` |
| Numeric Types: | `int`, `float`, `complex` |
| Sequence Types: | `list`, `tuple`, `range` |
| Mapping Type: | `dict` |
| Set Types: | `set`, `frozenset` |
| Boolean Type: | `bool` |
| Binary Types: | `bytes`, `bytearray`, `memoryview` |
| None Type: | `NoneType` |

s.upper()
s.lower()
s.split()
s.strip()
s.replace()

**FORMATTING**

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

**SNIPPET**

```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

We are the so-called "Vikings" from the north.

```
# Python3 program to show the
# working of upper() function
text = 'geeKs For geEkS'

# upper() function to convert
# string to upper case
print("\nConverted String:")
print(text.upper())
```

```
# lower() function to convert
# string to lower case
print("\nConverted String:")
print(text.lower())

# converts the first character to
# upper case and rest to lower case
print("\nConverted String:")
print(text.title())

#swaps the case of all characters in the string
# upper case character to lowercase and viceversa
print("\nConverted String:")
print(text.swapcase())

# convert the first character of a string to uppercase
print("\nConverted String:")
print(text.capitalize())

# original string never changes
print("\nOriginal String")
print(text)
```

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
False
False
False
False
False
False
False
```

```
print(bool("abc"))
print(bool(123))
print(bool(["apple", "cherry", "banana"]))
```

**True**
**True**
**True**

**thislist = ["apple", "banana", "cherry"]**
**thislist.insert(2, "watermelon")**
**print(thislist)**

**LAMBDA**

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

---

# Why Use Lambda Functions?

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n
```

---

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
---------------------------------------------------------------------
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
---------------------------------------------------------------------
```

# Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

# The __init__() Function

All classes have a function called __init__(), which is always executed when the class is being initiated.

The __init__() function is called automatically every time the class is being used to create a new object.

```
class Person:
  def __init__(self, name, age):
    self.name = name
     self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

# The __str__() Function

The __str__() function controls what should be returned when the class object is represented as a string.

```
class Person:
  def __init__(self, name, age):
    self.name = name
     self.age = age

p1 = Person("John", 36)

print(p1)
```

# The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

# Delete Object Properties

You can delete properties on objects by using the `del` keyword:

```
del p1.age
```

# Delete Objects

```
del p1
```

---

**str()**
**int()**
**float()**
**print()**
**type()**