Python

Hello, Python!

Python was named for the British comedy troupe Monty Python, so we'll make our first Python program a homage to their skit about Spam).

Just for fun, try reading over the code below and predicting what it's going to do when run. (If you have no idea, that's fine!)

Then click the "output" button to see the results of our program.

Developed by - Guido van Rossum, Dutch Programmer, 20 Feb 1991

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Quotient of `a` and `b`, removing fractional parts
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`

order of calculation - PEMDAS (left to right direction)

commanly used types - str, int, float, Boolean.

```
spam_amount = 0
print(spam_amount)

type(spam_amount)
# 0 + 4 = 4; spam_amount = 4
```

```
spam_amount = spam_amount + 4
print(spam_amount)

if spam_amount > 0:
    print("But I don't want ANY spam!")

# prints spam 4 times
viking_song = "Spam " * spam_amount
print(viking_song)
```

```
0
4
But I don't want ANY spam!
Spam Spam Spam Spam
```

1.Numbers and Arithmetics in Python

python - type()

```
spam_amount = 0
type(spam_amount)
```

```
int
```

```
type(19.95)
```

```
float
```

```
print(5/2)
print(6/2)
```

```
2.5
3.0
```

```
print(5//2)
print(6//2)
```

```
2
3
```

Order of Operation

```
8 - 3 + 2
# order of operation
```

```
# 5 + 2
# 7
```

```
-3 + 4 * 2
# order of operation
# -3 + 8
# 5
```

```
5
```

```
hat_height_cm = 25
my_height_cm = 190

# wrong formula
total_height_meters = hat_height_cm + my_height_cm / 100
print("Height in meters =", total_height_meters, "?")

# correct formula
total_height_meters = (hat_height_cm + my_height_cm) / 100
print("Height in meters =", total_height_meters)
```

```
Height in meters = 26.9 ?
Height in meters = 2.15
```

python - min() and max()

```
# min = min(1,2,3)
# print(min)

# max = max(3,4,5,722,44,56)
# print(max)
```

python - abs() -> absolute value (+ve)

```
print(abs(32))
print(abs(-32))
```

```
32
32
```

Type conversion

```
print(float(10))
print(int(3.33))

print(int('807') + 1)
```

```
10.0
3
808
```

```
f = 10.22
print(type(f))

int_num = int(f) # or int(10.22)
print(type(int_num))
```

```
<class 'float'>
<class 'int'>
```

Excercise - Numbers and Arithmetics in Python

Q1. Create a string and print its type and value

```
color = 'blue'
print(type(color))
print(color)
```

```
<class 'str'>
blue
```

Q2. find area and perimeter of circle

Concate -> type conversion

```
pi = 3.14159 # approximate
diameter = 3

radius = float(diameter/2)
print('radius is: ' + str(radius))

area = pi * (radius**2)
print('area is: ' + str(area))
```

```
radius is: 1.5
area is: 7.0685775
```

Q3. Swap values of a and b

```
a = [1, 2, 3]
b = [3, 2, 1]

tmp = a
a = b
b = tmp

print("a = " + str(a))
print("b = " + str(b))
```

```
a = [3, 2, 1]
b = [1, 2, 3]
```

Q4. divisibilty

Alice, Bob and Carol have agreed to pool their Halloween candy and split it evenly among themselves. For the sake of their friendship, any candies left over will be smashed. For example, if they collectively bring home 91 candies, they'll take 30 each and smash 1.

Write an arithmetic expression below to calculate how many candies they must smash for a given haul.

```
# Variables representing the number of candies collected by alice, bob, and carol
alice_candies = 121
bob_candies = 77
carol_candies = 109

# Your code goes here! Replace the right-hand side of this assignment with an expression
total = alice_candies + bob_candies + carol_candies
if (total % 3) == 0:
    each_gets = total/3
    print("each gets: " + str(each_gets))
else:
    x = total % 3
    total1 = total - x
    each_gets = total1 / 3
    print("each gets: " + str(each_gets))
```

```
each gets: 102.0
```

2.Functions and Getting Help

help() - help(print) -> shows info about print function and parameters it can take

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Functions

function defination

Functions can be definded using def keyword Functions take parameters as input (paramters can have default value or no value, values of parameter we be replaced by user inputed value function without return gives None

Find least deference

```
# def least_dif(a, b, c):
#   dif1 = abs(a - b)
#   dif2 = abs(b - c)
#   dif3 = abs(a - c)
#   minval = min(dif1, dif2, dif3)
#   return minval

# function_returned_value = least_dif(123, 543534, 6666566)
# print(function_returned_value)
```

```
def greet(who='enter a name'):
  print("Hello, " + who)

greet('simon')
```

```
Hello, simon
```

```
print(1, 2, 3, sep='<')
```

```
1<2<3
```

```
def multiply_by_five(x):
  # multiply number with 5
```

```
  return 5*x

def mod_5(x):
  # return remender if divided by 5
  return x % 5

def call(function, arg):
  # call function using function and apply argument
  return function(arg)

def sq_fn(function, arg):
  # call function using function and apply in function and apply argument
  return function(function(arg))

print(multiply_by_five(44),
      mod_5(56),
      call(multiply_by_five, 48),
      call(mod_5, 554),
      sq_fn(mod_5, 33),
      sq_fn(multiply_by_five, 445),
      sep='\n')
```

```
220
1
240
4
3
11125
```

Excercise - Function and Getting Help

Q1. Round to two numbers -> round()

```
def round_two_nums(num):
  number2 = round(num, 2)
  return number2

y = round_two_nums(3.543545)
print(y)
```

```
3.54
```

Q2. round()

```
x = round(3.12464253, -2)
y = round(31246.4253, -2)
z = round(312464253, -2)
print(x, y, z, sep='\n')
```

```
0.0
31200.0
312464300
```

Q3. remender find candies to smash

```
# candies to smash

def candies_to_smash(candies, friends):
  to_smash = candies % friends
  return to_smash
x = candies_to_smash(453, 5)
print(x)
```

```
3
```

3.Booleans and Conditionals

```
def age_of_prez(age):
  return age >= 35

print('can run for prez: ', age_of_prez(4))
```

```
can run for prez:  False
```

```
3.0 == 3
```

```
True
```

even or odd

```
def is_odd(n):
  return (n % 2) != 0

print('number is odd: ', is_odd(3))
```

```
number is odd:  True
```

```
def can_be_prez(age, citizen):
  return citizen and (age >= 35)

print('eligible or not: ', can_be_prez(33, True))
print('eligible or not: ', can_be_prez(35, True))
print('eligible or not: ', can_be_prez(35, False))
```

```
eligible or not:  False
eligible or not:  True
eligible or not:  False
```

```
True or True and False
```

```
True
```

Umbrela problem

```
def prep_for_weather(have_umbrela, rain_level, have_hood, is_workday):
  good_to_go = (
      have_umbrela
      or ((rain_level < 5) and have_hood)
      or (not ((rain_level > 0) and is_workday))
  )
  return good_to_go

print('prep_for_weather or not prep: ', prep_for_weather(True, 4, False, True))
```

```
prep_for_weather or not prep:  True
```

Conditionals

Positive or negative number

```
def check_negative_num(x):
  if x == 0:
    print('num is zero')
  elif x < 0:
    print('num is negative')
  else:
    print('num is positive')

check_negative_num(33459)
check_negative_num(-4474)
check_negative_num(0)
```

```
num is positive
num is negative
num is zero
```

Excercise - Booleans and Conditionals

Q1. sign() -> Returns 0 if num is 0, 1 if num is +ve and -1 if num is -ve

```
def sign(num):
  if num == 0:
    sign = 0
  elif num < 0:
    sign = -1
  elif num > 0:
    sign = 1
  else:
    sign = 'enter valud num'
  return sign

print('sign of num is ', sign(4))
```

```
sign of num is  1
```

Q2. splitting cadies grammer

```
def to_smash(total_candies):
  ans = print('Splitting', total_candies, 'candy' if total_candies == 1 else 'candies')
  return ans

to_smash(2)
to_smash(41)
to_smash(1)
```

```
Splitting 2 candies
Splitting 41 candies
Splitting 1 candy
```

Q3. check negative one line code

```
def negative(x):
  return True if x < 0 else False

print('is negative?: ', negative(3))
print('is negative?: ', negative(-3))
```

```
is negative?:  False
is negative?:  True
```

Q4. Hot dog toppings mustard, ketchup, onion

```
def onionless(ketchup, mustard, onion):
    # no onion
    return ketchup and mustard and (not onion)

print(onionless(True, True, False))
print(onionless(True, True, True))
print(onionless(False, False, False))
print(onionless(True, False, False))
print(onionless(True, False, True))
```

```
True
False
False
False
False
```

```
def want_all_toppings(ketchup, mustard, onion):
  # all toppings
  return ketchup and mustard and onion

print(want_all_toppings(True, False, True))
print(want_all_toppings(True, True, True))
```

```
False
True
```

```
def want_all_toppings(ketchup, mustard, onion):
  # no toppings
  return not(ketchup or mustard or onion)

print(want_all_toppings(True, True, True))
print(want_all_toppings(False, False, False))
print(want_all_toppings(True, False, True))
```

```
False
True
False
```

```
def one_sauce(ketchup, mustard, onion):
  # only one sauce either ketchup ot mustard with onion
  return (ketchup and not mustard) or (mustard and not ketchup)

print(one_sauce(True, False, True))
print(one_sauce(True, False, False))
print(one_sauce(False, True, True))
print(one_sauce(True, True, True))
print(one_sauce(True, True, False))
```

```
True
True
True
False
False
```

```
def only_one(ketchup, mustard, onion):
  # only one topping
  return (int(ketchup) + int(mustard) + int(onion)) == 1

print(only_one(True, False, True))
print(only_one(False, True, False))
```

```
False
True
```

Q5. Blackjack Should Hit or not

```

```

4.List and Tuples

Lists

```
primes = [2,3,5,7]
primes
```

```
[2, 3, 5, 7]
```

```
string_in_lists = ['xyz', 'abc', 'mnp']
string_in_lists
```

```
['xyz', 'abc', 'mnp']
```

```
multitype_lists = ['abc', 1, True, 1.0]
multitype_lists
```

```
['abc', 1, True, 1.0]
```

```
nested_lists = [[1,2,3],['a','b','c'],[1,'a',True]]
nested_lists
```

```
[[1, 2, 3], ['a', 'b', 'c'], [1, 'a', True]]
```

```
nested_lists1 = [[1000,2000,3000],
                 ['abc','xyz','lmn'],
                 [100,'x',False]]
nested_lists1
```

```
[[1000, 2000, 3000], ['abc', 'xyz', 'lmn'], [100, 'x', False]]
```

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
planets
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

Lists Indexing

```
planets[1]
```

```
'Venus'
```

```
planets[0]
```

```
'Mercury'
```

```
planets[-1]
```

```
'Neptune'
```

```
planets[-2]
```

```
'Uranus'
```

Lists Slicing

```
planets[0:3]
```

```
['Mercury', 'Venus', 'Earth']
```

```
planets[:3]
```

```
['Mercury', 'Venus', 'Earth']
```

```
planets[3:]
```

```
['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

```
planets[1:-1]
```

```
['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus']
```

```
planets[1:]
```

```
['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

```
planets[-3:]
```

```
['Saturn', 'Uranus', 'Neptune']
```

Changing Lists

```
planets[3] = 'Malacandra'
planets
```

```
['Mercury',
 'Venus',
 'Earth',
 'Malacandra',
 'Jupiter',
 'Saturn',
 'Uranus',
 'Neptune']
```

```
planets[:3] = ['mur', 'vee', 'ur']
planets
```

```
['mur', 'vee', 'ur', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

```
planets[:4]
```

```
['mur', 'vee', 'ur', 'Malacandra']
```

```
planets[:4] = ['Mercury', 'Venus', 'Earth', 'Mars']
planets
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

List Functions

```
len(planets)
```

```
8
```

```
sorted(planets)
```

```
['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']
```

```
primes = [2,3,5,7]
sum(primes)
```

```
17
```

```
# max(primes)
```

Interlude: objects

I've used the term 'object' a lot so far - you may have even read that everything in Python is an object. What does that mean?

In short, objects carry some things around with them. You access that stuff using Python's dot syntax.

```
x = 12
print(x.imag) # imaginary part associated with 12

c = 12 + 3j # creating complex number
print(c.imag)
```

```
0
3.0
```

Method

The things an object carries around can also include functions. A function attached to an object is called a method

```
x.bit_length
```

```
<function int.bit_length>
```

```
x.bit_length()
```

```
4
```

```
help(x.bit_length)
```

```
Help on built-in function bit_length:

bit_length() method of builtins.int instance
    Number of bits necessary to represent self in binary.

    >>> bin(37)
    '0b100101'
    >>> (37).bit_length()
    6
```

List Method

```
planets.append('pluto')
```

```
planets
```

```
['Mercury',
 'Venus',
 'Earth',
 'Mars',
 'Jupiter',
 'Saturn',
 'Uranus',
 'Neptune',
 'pluto']
```

```
planets.pop()
```

```
'pluto'
```

```
planets
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

Lists Searching

```
planets.index('Earth')
```

```
2
```

```
# planets.index('pluto')
```

```
"Earth" in planets
```

```
True
```

```
"pluto" in planets
```

```
False
```

Tuples

```
t = (1,2,3)
t
```

```
(1, 2, 3)
```

```
t1 = (1, 'y', True)
t1
```

```
(1, 'y', True)
```

```
# valies cannot be changed or assinged
# t[100] = 44
```

Tuples are often used for functions that have multiple return values.

```
x = 0.125
x.as_integer_ratio()
```

```
(1, 8)
```

```
num, denom = x.as_integer_ratio()
num, denom
```

```
(1, 8)
```

```
a = 1
b = 2
a, b = b, a
a, b
```

```
(2, 1)
```

Excercise - Lists and Tuples

Q1. Return 2nd last element of a list

```
def select_second(L):
    if len(L) < 1:
      return None
    return L[1]

listss = []
print(select_second(listss))
```

```
None
```

Q2. return 2nd element of 2nd last list (nested lists)

```
def access_nested_loops(teams):
  return teams[-2][1]

teams = [[1,2,3],[11,22,33],['y','a','z','q']]
print(access_nested_loops(teams))
```

```
22
```

Q3. swap 1st and last element of list

```
def swap_first_last(racers):
  tmp = racers[0]
  racers[0] = racers[-1]
  racers[-1] = tmp
  return racers

racers = [1,2,3,4]
print(swap_first_last(racers))
```

```
[4, 2, 3, 1]
```

Q4. len of lists

```
a = [1, 2, 3]
print(a)
print(len(a))
b = [1, [2, 3]]
print(b)
print(len(b))
c = []
print(c)
print(len(c))
d = [1, 2, 3][1:]
print(d)
print(len(d))
```

```
[1, 2, 3]
3
[1, [2, 3]]
2
[]
0
[2, 3]
2
```

Q5. Ignore last element and 1st half of the List

```
party_attendees = ['Adela', 'Fleda', 'Owen', 'May', 'Mona', 'Gilbert', 'Ford']

def late(arrivals, name):
  order = arrivals.index(name)
  return order >= len(arrivals) / 2 and order != len(arrivals) - 1

print(late(party_attendees, 'Adela'))
```

```
False
```

```
order = party_attendees.index('Owen')
order
```

```
2
```

5.Loops and Lists comprehensions

For Loop

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
planets
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

```
for planet in planets:
  print(planet, end=' ')
```

```
Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune
```

```
multiplicands = (2, 2, 2, 3, 3, 5)
product = 1

for multi in multiplicands:
  product = multi*product

product
```

```
360
```

```
s = 'steganograpHy is the practicE of conceaLing a file, message, image, or video within a

for char in s:
  if char.isupper():
    print(char, end='')
```

```
HELLO
```

```
for i in range(5):
  print('this is number ', i)
```

```
this is number  0
this is number  1
this is number  2
this is number  3
this is number  4
```

While Loop

```
i = 0
while i < 10:
  print(i, end=' ')
  i += 1
```

```
0 1 2 3 4 5 6 7 8 9
```

List Comprehensions

```
squares = [n**2 for n in range(10)]
squares
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
squares = []
for i in range(10):
  i = i*i
  squares.append(i)

squares
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
short_planets = [planet for planet in planets if len(planet) < 6]
short_planets
```

```
['Venus', 'Earth', 'Mars']
```

```
loud_planets = [planet.upper() + '!' for planet in planets if len(planet) < 6]
loud_planets
```

```
['VENUS!', 'EARTH!', 'MARS!']
```

```
['lets replace all planet name with this sentence' for planet in planets]
```

```
['lets replace all planet name with this sentence',
 'lets replace all planet name with this sentence',
 'lets replace all planet name with this sentence',
```

```
        'lets replace all planet name with this sentence',
        'lets replace all planet name with this sentence',
        'lets replace all planet name with this sentence',
        'lets replace all planet name with this sentence',
        'lets replace all planet name with this sentence']
```

```
def count_neg(n):
  i = 0
  for a in n:
    if a < 0:
      i += 1
  return i

print(count_neg([-2,44,63,-64,-1]))
```

```
3
```

```
def count_neg(n):
  return len([i for a in n if a <0])

print(count_neg([23,-1,-55,33]))
```

```
2
```

```
def count_neg(n):
  return sum([a < 0 for a in n])

print(count_neg([22,-656,-66,77434]))
```

```
2
```

Excercise - Loops and List Comprehensions

Q1. divisibility of 7

```
def has_lucky_num(nums):
  for num in nums:
    if num % 7 == 0:
      return True
    return False

print(has_lucky_num([34,55,11,356]))
```

```
False
```

Q2. if element of list is greater than given number

```
def element_greater_than(L, gr):
  return [l > gr for l in L]

print(element_greater_than([2,4,7,3,8,55], 5))
```

```
[False, False, True, False, True, True]
```

Q3. Check if two consicative values are eqaul

```
def iterate_over_list(meals):
  for i in range(len(meals)-1):
    if meals[i] == meals[i+1]:
      return True
    return False

print(iterate_over_list([2,6,33,4,66,3]))
```

```
False
```

Q4. Slot Machine

```

```

5.Strings and Dicts

```
planets = 'pluto'
planets[0]
```

```
'p'
```

```
planets[-3:]
```

```
'uto'
```

```
len(planets)
```

```
5
```

```
[char.upper() + '!' for char in planets]
```

```
['P!', 'L!', 'U!', 'T!', 'O!']
```

```
planets.upper()
```

```
'PLUTO'
```

```
planets.lower()
```

```
'pluto'
```

```
planets.index('o')
```

```
4
```

```
planets.startswith('p')
```

```
True
```

```
planets.endswith('t')
```

```
False
```

```
for word in planets:
  print(word.split(sep=', '))
```

```
['p']
['l']
['u']
['t']
['o']
```

```
day = '1945-23-6'
yr, mn, dy = day.split('-')
yr, mn, dy
```

```
('1945', '23', '6')
```

```
date = '/'.join([yr, mn, dy])
date
```

```
'1945/23/6'
```

```
"{}, is the {}th planet".format('pluto', 9)
```

```
'pluto, is the 9th planet'
```

Dicts

```
numbers = {'one':1, 'two':2, 'three':3}
numbers
```

```
{'one': 1, 'three': 3, 'two': 2}
```

```
numbers['one']
```

```
1
```

```
numbers['three'] = 'earth'
```

```
numbers
```

```
{'one': 1, 'three': 'earth', 'two': 2}
```

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

initials = {planet: planet[0] for planet in planets}
initials
```

```
{'Earth': 'E',
 'Jupiter': 'J',
 'Mars': 'M',
 'Mercury': 'M',
 'Neptune': 'N',
 'Saturn': 'S',
 'Uranus': 'U',
 'Venus': 'V'}
```

```
'Saturn' in initials
```

```
True
```

```
'V' in initials.values()
```

```
True
```

```
for num in numbers:
  print("{} = {}".format(num, numbers[num]))
```

```
one = 1
two = 2
three = earth
```

```
sorted(initials.values())
```

```
['E', 'J', 'M', 'M', 'N', 'S', 'U', 'V']
```

Excercise - Strings and Dicts

Q1. check if string is exactly of length 5 and it consist of numeric values

```
def zip_code_checker(zipcode):
  if zipcode.isdigit() and len(zipcode) == 5:
    return True
  else:
    return False

print(zip_code_checker('38485'))
```

```
 True
```

Q2. Search a word in given list and its element

```
doc_list = ["The Learn Python Challenge Casino.", "They bought a car", "Casinoville"]
doc_list
```

```
['The Learn Python Challenge Casino.', 'They bought a car', 'Casinoville']
```

```
def word_search(doc_list, keyword):
    # list to hold the indices of matching documents
    indices = []
    # Iterate through the indices (i) and elements (doc) of documents
    for i, doc in enumerate(doc_list):
        # Split the string doc into a list of words (according to whitespace)
        tokens = doc.split()
        # Make a transformed list where we 'normalize' each word to facilitate matching.
        # Periods and commas are removed from the end of each word, and it's set to all lo
        normalized = [token.rstrip('.,').lower() for token in tokens]
        # Is there a match? If so, update the list of matching indices.
        if keyword.lower() in normalized:
            indices.append(i)
    return indices
```

```
def multi_word_search(doc_list, keywords):
    keyword_to_indices = {}
    for keyword in keywords:
        keyword_to_indices[keyword] = word_search(documents, keyword)
    return keyword_to_indices
```

```

```

7.Working with Libraries

Import

```
import numpy
import pandas
```

Import as syntax

```
import numpy as np
import pandas as pd
```

Import all modules

```
from numpy import *
from pandas import *
```

Import Submodules

```
from numpy import random
from pandas import DataFrame
```

```
import math
```

```
print(math.pi)
```

```
3.141592653589793
```

```
print(math.log(32,55))
```

```
0.8648484522253854
```

```
from numpy import asarray, random
```

```
rolls = numpy.random.randint(low=1, high=6, size=10)
rolls
```

```
array([4, 5, 1, 2, 1, 4, 2, 5, 3, 3])
```

```
rolls.mean()
```

```
3.0
```

```
rolls.tolist()
```

```
[4, 5, 1, 2, 1, 4, 2, 5, 3, 3]
```

```
rolls + 10
```

```
array([14, 15, 11, 12, 11, 14, 12, 15, 13, 13])
```

Operator overloading

```
opelist = [1, 3, 4, 1, 5]
'''

> opelist + 10

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-b2f166bf1fce> in <module>()
      1 opelist = [1, 3, 4, 1, 5]
----> 2 opelist + 10

TypeError: can only concatenate list (not "int") to list
'''
```

```
'\n\n> opelist + 10\n\n---------------------------------------------------------------
---------------\nTypeError                                 Traceback (most recent c
all last)\n<ipython-input-19-b2f166bf1fce> in <module>()\n      1 opelist = [1, 3,
4, 1, 5]\n----> 2 opelist + 10\n\nTypeError: can only concatenate list (not "int")
```

```
rolls = array(opelist) + 10
rolls
```

```
<IntegerArray>
[11, 13, 14, 11, 15]
Length: 5, dtype: Int64
```

```
rolls <= 3
```

```
<BooleanArray>
[False, False, False, False, False]
Length: 5, dtype: boolean
```