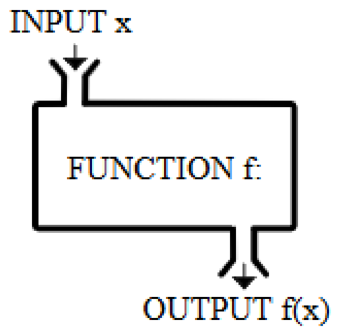
Function

- A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- As you already know, Python gives you many **built-in functions** like `print(),` etc. but you can also create your own functions. These functions are called ***user-defined functions.***

Function

- Input: arguments
- Output: return value



Python built in function

In the context of programming, a function is a named sequence of statements that performs a computation. When you define a function, you specify the name and the sequence of statements. Later, you can “call” the function by name. We have already seen one example of a function call :

```
>>>type(32)
<type 'int' >
```

The name of the function is type . The expression in parentheses is called the arguments of function. The result, for this function, is the type of the argument.

Math functions

Python has a **math module** that provides most of the familiar mathematical functions. A module is a file that contains a collection of related functions. Before we can use the module, we have to import the functions from the Python library:

```
import math
```

The module object contains the functions and variables defined in the module. To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period). This format is called dot notation.

```
x=3.1416
math.ceil(x)
```

Some Python built in math function

Method	Description
`math.ceil()`	Rounds a number up to the nearest integer
`math.cos()`	Returns the cosine of a number
`math.degrees()`	Converts an angle from radians to degrees
`math.factorial()`	Returns the factorial of a number
`math.floor()`	Rounds a number down to the nearest integer
`math.radians()`	Converts a degree value into radians
`math.sin()`	Returns the sine of a number
`math.pi`	Use the pi value of 3.141592653589793238
`math.sqrt()`	Return the squared root of a number
`math.pow(base,exponent)`	Return *base* to the power of *exponent*.

Math functions

Example) The circumference of a circle given the radius.
Formula circumference = 2*radius*pi

```
import math
radius = int(input("Enter a radius: "))
circumference = 2*radius*math.pi
circumference=round(circumference,2)
print("The circumference with radius %s is %s" %(radius,
circumference))
```

The expression **math.pi** gets the variable pi from the math module. The value of this variable is an approximation of pi, accurate to about 15 digits.

Math functions

Example) find the area of a circle given the radius of the circle. Formula area = radius^2*pi

```
import math
radius = int(input("Enter a radius: "))
area = math.pow(radius,2)*math.pi
area = math.floor(area)
print("The area with radius %s is %s" %(radius, area))
```

Python number method **math.pow()** returns *radius* to the power of *2*.

Random Number

Import library random ➔ `import random`

random() – Return the next random floating point number in the range [0.0, 1.0).

Repetitively calling rand function will issue a sequence of random numbers based upon the original seed.

By updating or randomizing the seed in some way, it is possible to generate different pseudorandom sequences.

```
import random
# Will return a number, float, between 0 and 10
print("uniform()", random.uniform(0,10))
# Will return an integer between 0 and 10
print("randint()", random.randint(0,10))
# Will return  number multiple of 5 between 0 and 101
print("randrange()", random.randrange(0,101,5))
# Will return a number between 0 and RAND_MAX
print("random()", random.random())
for x in range(10):
  print(random.randint(1,101))
```

Example) random numbers. Randomly pick a color from a list
color =['red','blue','green']
randomIndex = random.randint(0,2)

Importing with from

Python provides two ways to import modules. If you import math , you get a module object named math. The module object contains constants like pi and functions. But if you try to access pi directly, you get an error.

```
from math import pi
print(pi)
```

```
from math import *
cos(pi)
```

The advantage of importing everything from the math module is that your code can be more concise. The disadvantage is that there might be conflicts between names defined in different modules, or between a name from a module and one of your variables.

User Defined Function

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

- A function definition specifies the name of a new function and the sequence of statements that execute when the function is called.
- In Python a function is defined using the **def** keyword:
- Syntax:

```
def name_function(parameters):
    statements
    …………
```

Example:

```
def my_function():
    print("Hello from a function")
```

Calling a Function

- To call a function, use the function name followed by parenthesis:

  Example:

```
def hello_function():
   print("Hello from a function")
hello_function()
```

Parameters

- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
- The following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

  Example:

```
def name(fname):
    print(" Welcome to the program: " + fname)
name("Tobias")
name("Peter")
name(input("Enter a name: "))
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without parameter, it uses the default value:

Example:

```
def country(c= "Norway"):
    print("I am from " + c)

country("Sweden")
country("India")
country()
country("Brazil")
```

Parameters and arguments

The arguments are assigned to variables called parameters.

The argument is evaluated before the function is called.

When you create a variable inside a function, it is local, which means that it only exists inside the function.

```
def sum(num1,num2):
    total = num1+num2
    return total

n1 = int(input("Enter num1: "))
n2 = int(input("Enter num2: "))
x = sum(n1,n2)
print("The sum of %s and %s is %s" %(n1,n2,x))
```

Return values

- Some functions produce results. Calling the function generates a value.
- All functions we have written so far are **void,** their return value is None.
- To let a function return a value, use the return statement:
- Example:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Return values

- As soon as a return statement executes, the function terminates without executing any subsequent statements. Code that appears after a return statement, or any other place the flow of execution can never reach, is called dead code .
- In a fruitful function, it is a good idea to ensure that every possible path through the program hits a return statement.

Keyword Arguments

- You can also send arguments with the *key* = *value* syntax.
- This way the order of the arguments does not matter.
- Example:

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Arbitrary Arguments

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a tuple of arguments, and can access the items accordingly:
- Example:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

Arbitrary Arguments

Passing an array

```
def children(*kids):
    n = len(kids[0])
    return kids
    # print("The second kid is %s amoung %s children" %(kids[1], n))
    print(n)

names = ['Emil','Tobias','Linus','Peter']
names1 = ['Emil','Tobias']
print(children(names, names1))
```

```
def function3(*kids):
    n = len(kids)
    print("The last kid is: " + kids[n-1])

function3("Tobias","Peter","Alex")
```

Incremental development

As you write larger functions, you might find yourself spending more time debugging.

To deal with increasingly complex programs, you might want to try a process called **incremental development**. The goal of incremental development is to avoid long debugging sessions by adding and testing only a small amount of code at a time.

```
def distance(x1, y1, x2, y2): #step1
  dx = x2 - x1
  dy = y2 - y1
  print('dx is' , dx)
  print('dy is' , dy)
  return 0.0

def distance(x1, y1, x2, y2): #step2
  dx = x2 - x1
  dy = y2 - y1
  dsquared = math.pow(x,2 )+ math.pow(dy,2)
  print('dsquared is: ' , dsquared)
  return 0.0

def distance(x1, y1, x2, y2): #step3
  dx = x2 - x1
  dy = y2 - y1
  dsquared = math.pow(x,2 )+
  math.pow(dy,2)
  result = math.sqrt(dsquared)
  return result
```

Composition

As you should expect by now, you can call one function from within another. This ability is called **composition** .

```
def sum(num1, num2):
    total = num1+num2
    return total

def number():
    n1 = int(input("Enter num1: "))
    n2 = int(input("Enter num2: "))
    t = sum(n1,n2)
    print("The sum of %s and %s is %s" %(n1,n2, t ))
```

Q: Please calculate the circle area of (1,2) to (-1,-5)

Boolean functions

Functions can return booleans, which is often convenient for hiding complicated tests inside functions.

```
def is_divisible(x, y):
     if x%y == 0 or y%x == 0:
          return True
     else:
          return False

is_divisible(6, 4)  ➔ False
is_divisible(6, 3) ➔ True
```

It is common to give boolean functions names that sound like yes/no questions.

The result of the == operator is a boolean, so we can write the function more concisely by returning it directly.

The pass Statement

- function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.
- Example:

```
def myfunction():
  pass
```

Why functions

1. Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read and debug.
2. Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
3. Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
4. Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.

Designing of functions

- There are many different views on what is consider a well design function.
- There are even arguments on why functions should be used at all, poorly designed functions will use up a lot of system resources, when the function is called.
- Each function should do one thing, achieve one task.
- Functions should be short, not more than X number of lines long
  - X being a number that the designer sees fit and it also depends on what the function needs to accomplish.
  - Think of it as writing a paragraph,as soon as you complete presenting the idea then you are done.

Exercise

Exercise 1) Write a function called **circleArea**() that takes an radius as argument, calculates the area of the circle, returns the area of the circle (circle area = pi $*r^2$). The radius is entered by the user.

Exercise 2) Write a function that simulate roll the dice that takes an number of roll as an argument. Within the function, for each round should display the random number between 1 and 6.